NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND RAPID PROTOTYPING OF FLIGHT CONTROL AND NAVIGATION SYSTEM FOR AN UNMANNED AERIAL VEHICLE

by

Bock-Aeng Lim

March 2002

Thesis Advisor:	Isaac I. Kaminer
Co-Advisor:	Oleg A. Yakimenko

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Design and Rapid Prototyping of Flight Control and Navigation System for an Unmanned Aerial Vehicle

5. FUNDING NUMBERS

6. AUTHOR(S) Bock-Aeng Lim

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)***

The work in this thesis is in support of a larger research effort to implement a cluster of autonomous airborne vehicles with the capability to conduct coordinated flight maneuver planning and to perform distributed sensor fusion. It seeks to design and implement an onboard flight control and navigation system for NPS FROG UAV, which will be used as the autonomous airborne vehicle for the research, using the newly marketed xPC Target Rapid Prototyping System from The Mathworks, Inc. Part I briefly introduces the aircraft and explains the necessity for an onboard computer for the UAV. Part II describes the construction of the miniature aircraft computer, INS/GPS and air data sensor integration implementation as well as the rapid prototyping process. Part III covers the process to create a 6DOF model for the aircraft and the design of the aircraft autopilot, while Part IV presents a vision-based navigation algorithm that can be implemented on the UAV to give it some form of autonomous flight trajectory planning capability. Ground test results showing successful onboard data integration are given to conclude this report.

14. SUBJECT TERMS Unmanned Aerial Vehicles, UAV, Autopilot, xPC Target, Rapid Prototyping, Guidance and Control	**15. NUMBER OF PAGES** 121
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 Rev.-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DESIGN AND RAPID PROTOTYPING OF FLIGHT CONTROL AND NAVIGATION SYSTEM FOR AN UNMANNED AERIAL VEHICLE

Bock-Aeng Lim
Major, Republic of Singapore Air Force
B.Eng.(Electrical), National University of Singapore, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING (AVIONICS)

from the

NAVAL POSTGRADUATE SCHOOL
March 2002

Author: Bock-Aeng Lim

Approved by: Isaac I. Kaminer, Thesis Advisor

Oleg A. Yakimenko, Co-Advisor

Max F. Platzer, Chairman
Department of Aeronautics and Astronautics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The work in this thesis is in support of a larger research effort to implement a cluster of autonomous airborne vehicles with the capability to conduct coordinated flight maneuver planning and to perform distributed sensor fusion. Specifically, it seeks to design and implement an onboard flight control and navigation system for NPS FROG UAV, which will be used as the autonomous airborne vehicle for the research, using the newly marketed xPC Target Rapid Prototyping System from The Mathworks, Inc. Part I briefly introduces the aircraft and explains the necessity for an onboard computer for the UAV. Part II describes the construction of the miniature aircraft computer, INS/GPS and air data sensor integration implementation as well as the rapid prototyping process. Part III covers the process to create a 6DOF model for the aircraft and the design of the aircraft autopilot, while Part IV presents a vision-based navigation algorithm that can be implemented on the UAV to give it some form of autonomous flight trajectory planning capability. Preliminary ground test results are presented in Part V to conclude this study.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I am grateful for the assistance I received from many people on this project. In particular, I would like to thank my thesis advisors Prof. Isaac Kaminer and Prof. Oleg Yakimenko for their directions and the latitude they allowed me to enjoy throughout this project. Their friendly nature created a conducive and informal learning environment for me. Their support and encouragement also made my job less stressful whenever I hit obstacles.

I am also grateful to Dr. Vladimir Dobohodkov, who is NRC Research Associate in the Aeronautical Department at NPS, for his technical assistance throughout this project. He has been a patient mentor and an outstanding colleague to work with. We spent many hours discussing problems encountered and exchanging ideas on how to solve them.

Thanks is also due to Jerry Lentz, our resident physicist and electronics guru in the department, who helped to package the computer and its power supply into a box that fitted nicely into the UAV, and later for his advice in the EMI investigations and to Don Meeks who had helped to disassemble, assemble and transport the UAV whenever I had to move it between to the Controls Lab and UAV Lab, for the machining work on the computer casing and for the tireless help he provided during the many ground tests. Thanks also to CDR Christopher Flood who had taught me much about the UAV and its control architecture when I understudied him while he was working on his thesis.

Most of all, a big thanks to my wife Irene who had been most understanding and supportive of the long hours I had to spend in the lab. She excused me from many household chores especially in the final quarter and even readily agreed to forego our precious Christmas break in the US just so that I could work on the project between the academic quarters.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

The Navy envisions that the battlefield of the future would include clusters of autonomous mobile agents equipped with a large number of sensors connected by wireless networks in a hostile and highly dynamic combat environment susceptible to hardware failures and jamming. As a result, changes in network topology and loss of connectivity between agents are expected. To allow the autonomous agents to continue their missions despite these changes in the network quality of service (QoS), they must have adequate ability to integrate data from as many working onboard sensors as possible to assess accurately the situational picture for decision making and for executing appropriate flight maneuvers.

The research question in support of such a battlefield setup is the execution of command, control and autonomous intelligent flight maneuver planning of a group of unmanned aerial vehicles and the construction of a distributed adaptive architecture for fusing sensor data over dynamically varying wireless networks [1]. The main thrust of this thesis is in line with the first research area mentioned above, i.e., to develop the basic framework to implement command and control (C^2) of a friendly cluster of autonomous unmanned aerial vehicles (UAVs) including algorithms for flight navigation and trajectory tracking in order to adopt certain flight profiles at various stages of the mission.

B. OBJECTIVES

This thesis seeks to implement an autonomous flight control and target approach guidance algorithm using the NPS FROG UAV as a test platform. The scope of work includes designing the autopilot for the aircraft, exploring suitable trajectory planning navigation algorithms and assembling an onboard computer to perform data fusion, flight control and guidance commands computation.

In addition, the timing of this effort coincided with the emergence of the xPC Target Rapid Prototyping System from The Mathworks, Inc. Such a system offers

enormous flexibility for implementing variants of the guidance and control algorithm or changes to the hardware architecture with significantly compressed design-to-flight-test time. Hence, a secondary objective of this project is to explore and accumulate expertise on this new tool, and apply it to the intended setup.

Details of both of these objectives will be covered in subsequent chapters. The rest of this chapter will serve as a lead-in by introducing the FROG UAV, which will act as the test aircraft, its instrumentation and the background on why an onboard computer is deemed necessary to implement the intended research objective.

C. THE AIRCRAFT

NPS's FROG UAV, shown in Figure I.1, has been the test bed for advanced control and airborne sensor projects at the Naval Postgraduate School [2,3]. It is manufactured by BAI Aerosystems as the BAI-TERN (Tactically Expendable Remote Navigator) and derives from the FOG-R variant of the BAI-TERN used by the US Army, hence the name 'FROG'. It is a small high wing monoplane with conventional elevator, rudder, ailerons and flaps, and uses servomotors designed for radio-controlled airplanes to drive the control surfaces. (Figure I.2). More details on its physical characteristics and engine are documented in Appendix A and in [4].

Previous control system projects made use of only very basic inertial sensing and a simple electromechanical autopilot in the aircraft. In the existing setup, the computer (shown in Figure I.3) which monitors flight data and computes aircraft control commands is located on the ground. Hence, raw flight data has to be downlinked from the aircraft to the computer via wireless serial modems for processing. Computed control inputs in turn are pulse-code modulated and re-transmitted back to the aircraft using a hand-held Futaba® remote transmitter, shown in Figure I.4, for the Futaba® receiver in the aircraft to interpret and output PWM commands to drive servos that controls the control surfaces.

Figure I.1 NPS FROG UAV

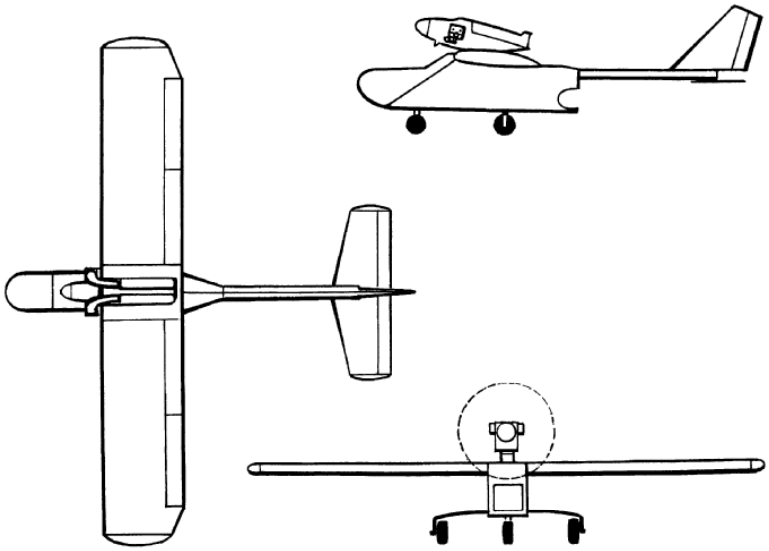


Figure I.2 FROG UAV 3 View Drawing.

Figure I.3 AC-104 Ground Control Computer

Figure I.4 Futaba® Transmitter, Receiver and Servos

D. CONTROL SETUP & ITS LIMITATIONS

The original setup introduced above imposes significant limitations on the complexity of the flight controller because of latency time to control commands generated by the computer on the ground. It also severely restricts the flight profiles that can be experimented due to operating range. An alternative method of controlling the servos to shorten the delay between computer command output and servo actuation was explored in [5]. It too could not fully meet desired specifications. The limitations are explained below.

1. Old Control Setup

In the original control scheme for the UAV illustrated in Figure I.5, the flight control computer was an AC-104 computer situated on the ground. Command signals from the AC-104 computer had to be converted to a pulse-code modulated (PCM) signal by a Futaba® radio controlled transmitter, which broadcasts them to the airplane. The Futaba® receiver in the FROG UAV decodes the PCM signal and generates pulse-width-modulated (PWM) commands for each of the control servos to control the aircraft. At the same time, in the feedback channel, sensor outputs are captured by auxiliary microprocessors, digitized and transmitted via wireless modem to the flight control computer on the ground for processing. Such a setup imposed severe controller restrictions due to the latency times for data downlink, control inputs computation on ground and the command uplink. In fact, the command uplink latency alone was measured to be approximately 170 ms in [4] and was found to generate unacceptable delay for any practical control frequency in the range of 20Hz to 40Hz.

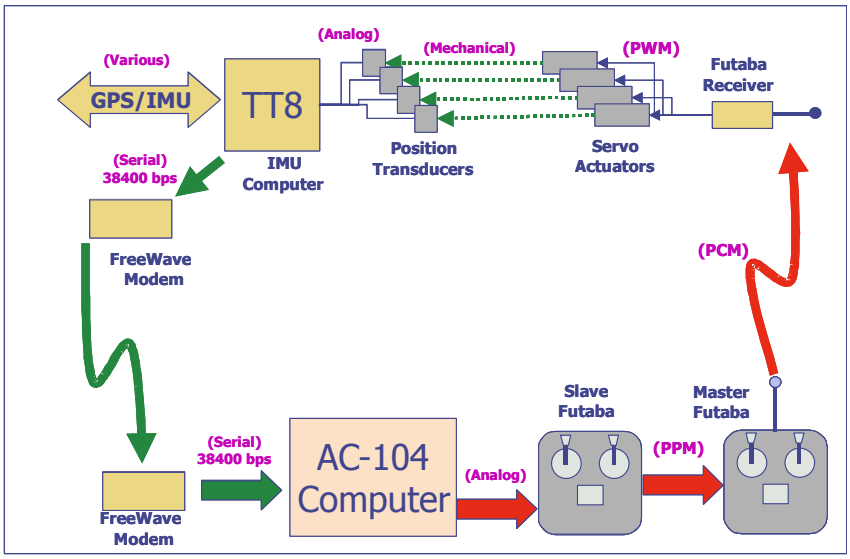


Figure I.5 Original Flight Control Setup (From [4])

2. Alternative Servo Command

The short range of the Futaba® remote controller and command path latency led to a feasibility study of using an alternate command uplink method by CDR Chris Flood and the author in [5]. In the proposed scheme, control commands were sent via serial modem from the AC-104 directly to a small onboard micro-controller which generates the PWM commands to the servo instead of routing the commands through the Futaba® remote control. Such a setup reduced the command latency to 76 msec. It was adequate to implement a workable controller but still placed severe restrictions on the controller performance.

The limitations mentioned above would have severe implications on the implementation of command and control for a cluster of UAVs. In order to support that objective, these constraints need to be overcome first. The proposed solution was to install a miniature computer in the UAV in order to minimize flight data and control commands transfer as well as to give the aircraft onboard computational capability to perform more sophisticated flight maneuver planning autonomously. This new hardware and control architecture implementation forms a significant part of this thesis work and is discussed in Chapter II.

E. PRIMARY INSTRUMENTS

The FROG UAV can be configured with a variety of instruments such as air-data sensors, an inertial measurement unit (IMU), a GPS receiver, an instrumented nose boom and even a digital camera. In this project, only sensors necessary for basic aircraft control, navigation and communication are installed as a basic configuration. These instruments are introduced in the following sub-sections.

1. Inertial Measurement Unit

A new altitude heading reference sensor was installed on the UAV for this project. This is the AHRS400CA-100 manufactured by Crossbow Technology, Inc shown in Figure I.6. The AHRS combines linear accelerometers, rotational rate sensors, and magnetometers to measure linear acceleration, angular velocity, and magnetic flux for all three orthogonal axes. It then utilizes a sophisticated Kalman filter algorithm to allow the unit to track orientation accurately through dynamic maneuvers in order to compute stabilized values of pitch, roll and true-magnetic heading. The Kalman filter will automatically adjust for changing dynamic conditions without any external user input. Hence, it can effectively function as the inertial measurement unit (IMU) for the aircraft.

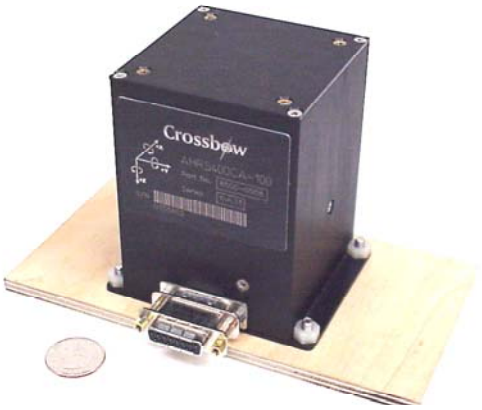


Figure I.6 Crossbow Technology's AHRS400CA-100

The AHRS400CA-100 can be operated in 3 sensor modes and the data available from each mode is shown in Table I.1. In all three sensor modes, the AHRS can measure linear acceleration up to ±2g and angular velocity up to ±100°/sec. For this project, the AHRS is operated in the Angle Mode in order to utilize its Kalman filtering capabilities.

In addition, data collection can be done in either continuous update or polled mode. A detailed comparison of data output rate in both modes was made in [4]. It was concluded that in continuous/angle mode, the AHRS outputs data at an alternating frequency of 69.7 Hz and 61.3 Hz, while in the polled mode the AHRS could not respond fast enough when polled at a fixed rate of 30 Hz. As such, the continuous mode was used in this project to take advantage of the higher data rate subject to an implementation that would accommodate a varying data rate.

The output data from the AHRS400CA-100 is provided in both digital and analog formats via a standard female DB-15 connector. The digital data is serially output via RS-232 interface at 38,400 bps and is the method used for reading data in this project. The data packet format and output data interpretation will be described in Section II when the data interface implementation between Crossbow AHRS and onboard computer is presented.

Angle Mode	Scaled Sensor Mode	Voltage Mode
Header (0xFF)	Header (0xFF)	Header (0xFF)
Roll Angle	Roll Angular Rate	Roll Gyro Voltage
Pitch Angle	Pitch Angular Rate	Pitch Gyro Voltage
Heading Angle	Yaw Angular Rate	Yaw Gyro Voltage
Roll Angular Rate	X-Axis Acceleration	X-Axis Acceleration Voltage
Pitch Angular Rate	Y-Axis Acceleration	Y-Axis Acceleration Voltage
Yaw Angular Rate	Z-Axis Acceleration	Z-Axis Acceleration Voltage
X-Axis Acceleration	X-Axis Magnetic Field	X-Axis Mag Sensor Voltage
Y-Axis Acceleration	Y-Axis Magnetic Field	Y-Axis Mag Sensor Voltage
Z-Axis Acceleration	Z-Axis Magnetic Field	Z-Axis Mag Sensor Voltage
X-Axis Magnetic Field	Temp Sensor Voltage	Temp Sensor Voltage
Y-Axis Magnetic Field	Time	Time
Z-Axis Magnetic Field	Checksum	Checksum
Temp Sensor Voltage		
Time		
Checksum		

Table I.1 Serial Outputs From AHRS400CA-100 In Various Sensor Modes

2. Global Positioning System Receiver

The GPS receiver used on the NPS Frog UAV is the Trimble Ag132 DGPS receiver as shown in Figure I.7. The Ag132 DGPS is a 12 channel L-band differential correction receiver that provides sub-meter accuracy. It combines a GPS receiver, a beacon differential receiver, and a satellite differential receiver in the same housing. These receivers use a combined antenna with a single antenna cable. The Ag132 is configured with two programmable RS-232 serial ports and outputs GPS data at 1, 5 or 10 Hz with latency of 10 msec in RS-232 serial ASCII format at baud rates up to 38,400 bps. All outputs conform to the National Marine Electronics Association (NMEA)-0183 data protocol. Among the various sentences in the GPS data stream shown in Table I.2, only some information in the $GPGGA and the $GPRMC sentences is relevant to our application and is extracted for use by the flight controller and guidance algorithm.

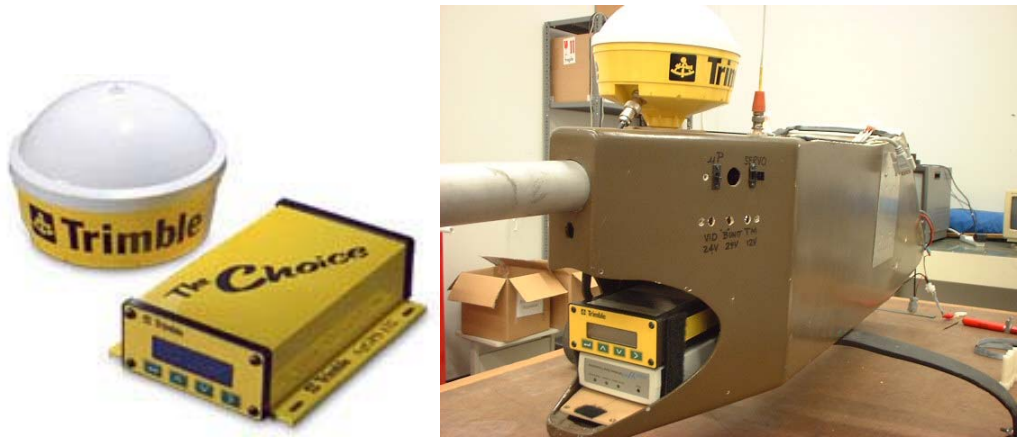


Figure I.7 Trimble Ag132 GPS Antenna and Receiver Mounted on FROG

Message	Contents
GGA	Time, position, and fix related data
GLL	Position fix, time of position fix, and status
GRS	GPS Range Residuals
GSA	GPS position fix mode, SVs used for navigation and DOP values
GST	GPS Pseudorange Noise Statistics
GSV	Number of SVs visible, PRN numbers, elevation, azimuth and SNR values
MSS	Signal strength, signal-to-noise ratio, beacon frequency, and beacon bit rate
RMC	UTC time, status, latitude, longitude, speed over ground (SOG), date, and magnetic variation of the position fix

VTG	Actual track made good and speed over ground
XTE Message	Cross-track error
ZDA	UTC time, day, month, and year, local zone number and local zone minutes.
PTNLDG Proprietary	Beacon channel strength, channel SNR, channel frequency, channel bit rate, channel number, channel tracking status, RTCM source, and channel performance indicator
PTNLEV Proprietary	Time, event number, and event line state for time-tagging change of state on a event input line.
PTNL,GGK	Time, Position, Position Type and DOP Values
PTNLID Proprietary	Receiver machine ID, product ID, major and minor release numbers, and firmware release date.
PTNLSM	Reference Station Number ID and the contents of the Special Message included in valid RTCM Type 16 records.

Table I.2 Trimble Ag132 Messages

3. Freewave® Radio Modems

The communication link between the UAV and the Host Computer on the ground is implemented with the DGR-115 RS-232 wireless modem shown in Figure I.8 from FreeWave® Technologies, Inc. The FreeWave modem uses frequency hopping spread spectrum technology and has a power output of 1/3 Watt. It is capable of communicating over a line of sight range of up to 20 miles, and supports data transmission at baud rates from 1200 bps to 115.2 Kbps. In addition, the FreeWave transceiver can operate in either point to point or point to multipoint modes, opening up the possibility to control of more than one aircraft in future. Its main specifications are shown in Table I.3

Figure I.8 DGR-115 Wireless Serial Modem (left) Mounted on UAV (right)

ITEM	SPECIFICATION
Range	20 Miles
RS232 Data Throughput	1200 Baud to 115.2 Kbaud
RS232 Interface	Asynchronous, Full duplex
System Gain	135 dB
Minimum Receiver Decode Level	-110 dBm @ 10-4 raw BER -108 dBm @ 10-6 raw BER
Operating Frequency	902 - 928 MHz
Modulation Type	Spread Spectrum, GFSK
Spreading Code	Frequency Hopping
Hop Patterns	15 (User Selectable)
Output Power	1/3 Watt (+25 dBm)
Error Detection	32 Bit CRC With Packet Retransmit
Antenna	3 Inch Whip Provided Non-standard SMA Connector Allows Use Of External Directional or Omni- Directional Antennas.
Power Requirements	10.5 - 18.0 VDC (Centre Positive)
Power Consumption	180 mA Transmit 100 mA Receive 120 mA Average
Connector	RS232 9 Pin Female, 9 Pin Male to 9 Pin Female Straight Through Cable Provided
Unit Address	Unique, Factory Preset
Operating Modes	Point to Point, Point to Multipoint Store and Forward Repeater
Operating Environment	-10° C to +50° C

Table I.3 Freewave® Radio Modem Specifications

4. Differential Pressure Sensor

A differential pressure sensor shown in Figure I.9 (Model: 144LU04DPC) from Sensortechnics was also installed in the FROG UAV to measure airspeed data. It is capable of measuring differential pressures up to 5 inches of water and outputs 0-5V depending on the pressure difference between the internal static pressure and that on the pitot probe mounted on the nose-fairing of the UAV shown in Figure I.10.

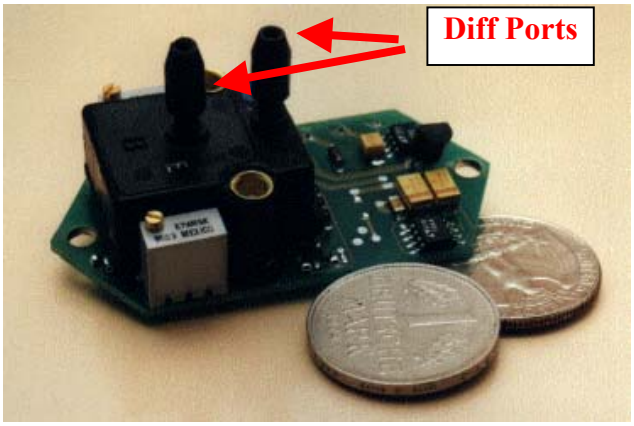


Figure I.9 Differential Pressure Transducer

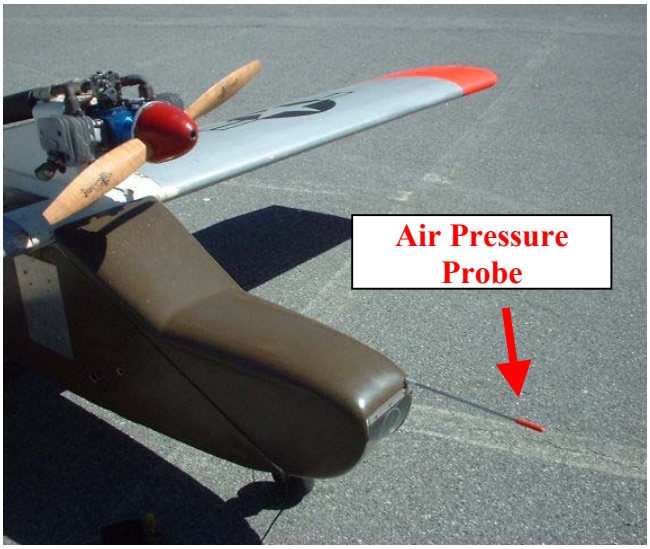


Figure I.10 Pitot Probe on UAV

II. RAPID PROTOTYPING OF NEW CONTROL SETUP

A. PROPOSED CONTROL SETUP

To overcome the constraints in the original setup for the conduct of the autonomous UAV cluster C^2 research, a new control setup shown in Figure II.1 was conceived. In this new setup, the onboard PC-104 computer would receive data from all airborne sensors and execute the flight control algorithm to stabilize and steer the aircraft. This reduces the latency time between data measurement to its use in computation, and to control commands output significantly. It also eliminates the need for flight data to be sampled at a high rate of 100Hz or more and transmitted to the computer on the ground for flight control computation, thereby reducing the bandwidth requirements on the serial modem tremendously. Instead, the onboard computer now only needs to download flight data to the host computer on the ground at a much lower 30 Hz to 40 Hz purely for data recording purposes, to still get fairly representative flight information for flight re-construction or parameter identification. As and when required, guidance commands for the autopilot can be sent to the flight control computer through the wireless serial modem link. This occurs at a much lower data rate.

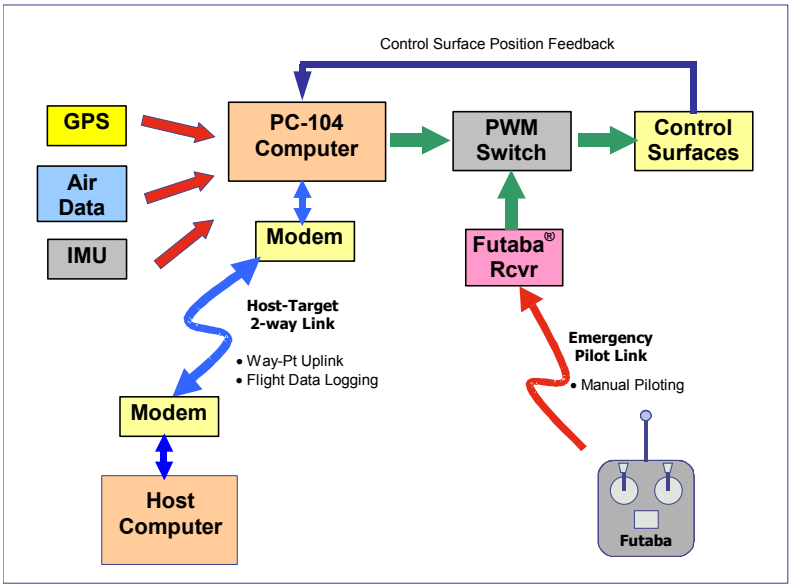


Figure II.1 New Control Setup

In an emergency, the onboard computer must quickly hand over the control authority of the aircraft back to the Futaba® remote control so that the pilot can regain control of the aircraft and pilot it from the ground. This is implemented with a PWM Switch that by default takes command from the Futaba® remote control unless the pilot grants control of the aircraft to the onboard computer. When the Futaba® remote control commands are modified to be sent through the serial modem, its range would increase correspondingly with the computer control to 20 miles, and this would allow future flight control research work at NPS to be conducted beyond the current 1.5 mile operating range.

The main constraint in the new setup is that the onboard computer must be powerful enough, and the data integration processes optimized sufficiently, to implement control at an acceptable rate. This consideration is constantly being monitored throughout this project and incorporated into the software drivers that are written to integrate sensor data.

B. THE RAPID PROTOTYPING SYSTEM

Implementation of the control setup was expedited using rapid prototyping techniques. A rapid prototyping system can be viewed as the complete set of hardware and software tools to implement and test the UAV's flight controller within a reasonably short period of development time. In this project, a deliberate decision was taken to shift from the previously used RealSim® Rapid Prototyping System by WindRiver, Inc, to the newly acquired xPC Rapid Prototyping System by The Mathworks, Inc, as the product and technical support for the former is gradually being phased out. To elaborate on the various components of the rapid prototyping system, it would be sub-divided into several sections– namely, the xPC Target rapid prototyping environment, the airborne computer, the data I/O hardware architecture.

1. xPC Target® Rapid Prototyping Package

xPC Target is a PC solution for prototyping, testing, and deploying real-time systems marketed by The Mathworks, Inc. The rapid prototyping process makes use of many toolboxes that are available with the popular Matlab simulation software to support implementation of control system simulation, data acquisition and real-time control. Computation and signal processing algorithms are first designed and tested as Simulink simulation models. xPC Target then makes use of the Real-Time Workshop® toolbox to convert the Simulink models into C-code, build real-time applications that can executed on any standard PC hardware and download into an assigned 'Target' PC. To interface the real-time application with hardware, users can make use of some of the I/O device drivers that comes with xPC Target to support commonly available I/O boards or develop their own device driver blocks by writing C-Mex S-functions in MATLAB.

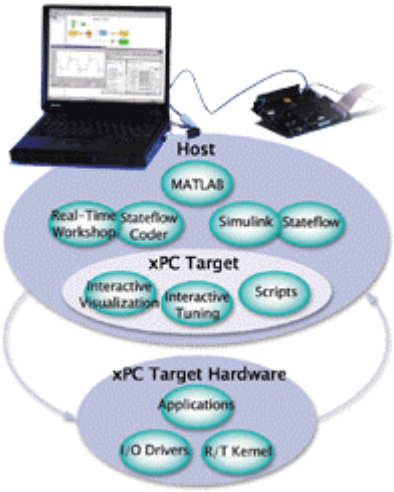


Figure II.2 xPC Target Setup

In a typical setup shown in Figure II.2, a Host computer makes use of Real-Time Workshop to build a real-time application based on Simulink models, I/O driver blocks and C-code S-functions created in MATLAB and downloads it to a Target computer via RS232 or TCP/IP using the xPC Target environment. The application in the target computer can be started, stopped, or its model parameters can be changed (called 'tuned' in xPC), and run-time data can be observed or recorded on the Host or Target PC. The

Target PC can be a common desktop PC, a PC/104, CompactPCI, industrial PC or Pentium Single Board Computer regardless of operating system because xPC uses its own real-time operating kernel. The Target PC is diskette bootable but can be made to boot-up internally and initiate the 'burnt in' application whenever it is reset if the optional xPC Target Embedded Option is installed.

The advantage of the xPC rapid prototyping system in this project is that it contains all the tools required to provide an integrated environment for control system design, software engineering, data acquisition and testing before it is finally implemented on the intended airborne computer. The software suite consists of the MATLAB package, Control System Toolbox, Simulink, Dials and Gauges Blockset, Real-Time Workshop and the xPC Target operating system. MATLAB and Control System Toolbox provide the capability to design, analysis and implement the flight controller. Simulink provides a graphical user interface (GUI) for construction of the controller, simulation and visualization. The Dials and Gauges Blockset contains a library of pre-designed blocks to facilitate design of a GUI for system parameters to be changed or values of variables to be displayed as the application is running. The Real-Time Workshop is an automatic code generator for the Simulink models. It converts the Simulink model and the S-Functions within it into C code which can then be compiled to create a stand-alone real-time executable program. Real-Time Workshop also schedules all the tasks to be carried out when the application is activated for the xPC operating system. Once compiled, the standalone executable code is suitable for the test-bed environment or for use in the embedded real-time system. The xPC real-time kernel sets up the environment in the Target PC to execute the application and maintain communications with the Host to allow parameter tuning as the application execution is in progress.

2. Host Computer

The Host Computer used in this project is a Pentium III notebook PC running Matlab and all the relevant rapid prototyping software toolboxes mentioned in the previous section and is situated on the ground. It has a standard serial COM1 port for the

Host-Target link implementation and a Ethernet network card for TCP/IP link-up with the Target PC which serves as a useful alternate link during development.

3. Target Airborne Computer

The Target PC installed on the NPS Frog UAV was re-configured from the AC-104 computer assembled by WindRiver Systems Inc. The AC-104 computer, shown previously in Figure I.3 was originally developed for WindRiver System's own RealSim® rapid prototyping system. The AC-104 uses a mixture of a PC/104 compact computer I/O card, Industrial Pack (IP) I/O modules and a small computer motherboard. It consists of an Intel Pentium MMX 233 MHz processor on an Advantech PCM-5862 motherboard configured with 16 MB of EDO RAM and a 4 MB flash disk for non-volatile storage. Basic I/O is provided by a PCI–SVGA display controller, two RS-232/422/485 serial ports, an enhanced parallel port, keyboard controller and a PCI based 10Base-T Ethernet connection.

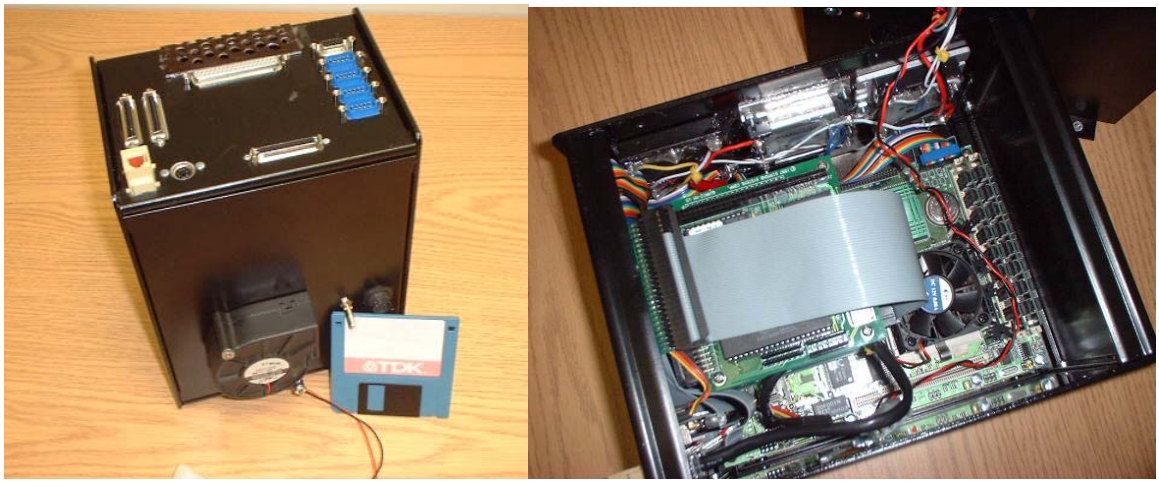


Figure II.3 New Miniature Airborne Computer

In the new miniature airborne computer shown in Figure II.3, the Diamond Systems Ruby-MM 12 bit D/A converter, the SBS GreenSprings Modular I/O Industry Pack IP-68322 data acquisition hardware control module and the IP-Serial board in the original setup were all removed. Only the Analogic AIM16 16-bit A/D converter, Flash Disk RAM card and motherboard were retained. A 3.5-inch floppy drive was added to

boot-up the xPC Target kernel and set of Quartz-MM-5 and Quartz-MM-10 counter/timer I/O boards by Diamond Systems was incorporated to get PWM signal generation and capture capability. The originally disabled COM3 and COM4 were also recovered with appropriate jumper settings on the motherboard and IRQ re-assignment. In addition, a set of switching power converters was built into the casing of the new computer to convert DC power from the aircraft's batteries to various voltages for the computer, cooling fan and onboard instruments. The resulting miniature computer fits snugly in the front compartment of the FROG UAV to give it additional forward center of gravity.

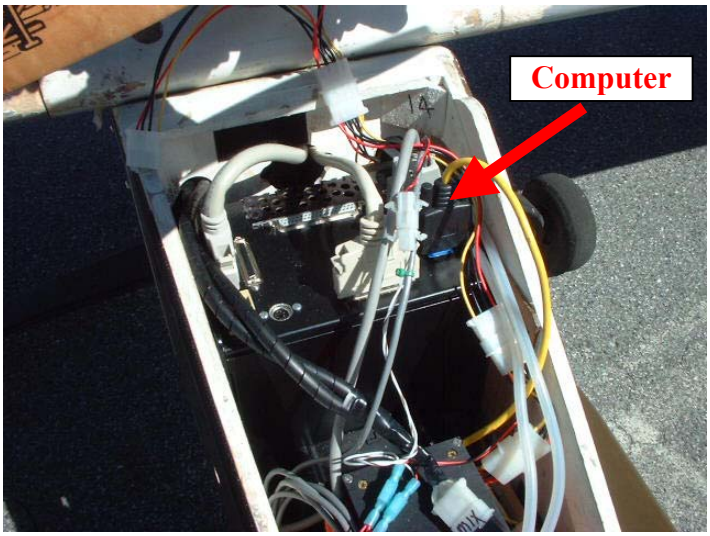


Figure II.4 Location of Miniature Computer on UAV

Signal I/O interface between the airborne target computer, the servos that drive the aircraft control surfaces and control surface position sensing potentiometers connect to the top of the computer as shown in Figure II.5. Within the target computer, the signal interface specifications for the I/O boards largely fall into three categories listed below and are tabulated for easy reference in the corresponding Appendices.

a. I/O Address – Appendix B.1

b. I/O Board Pin Out Connection and Usage – Appendix B.2

c. Interrupt Routine (IRQ) Assignment – Appendix B.3

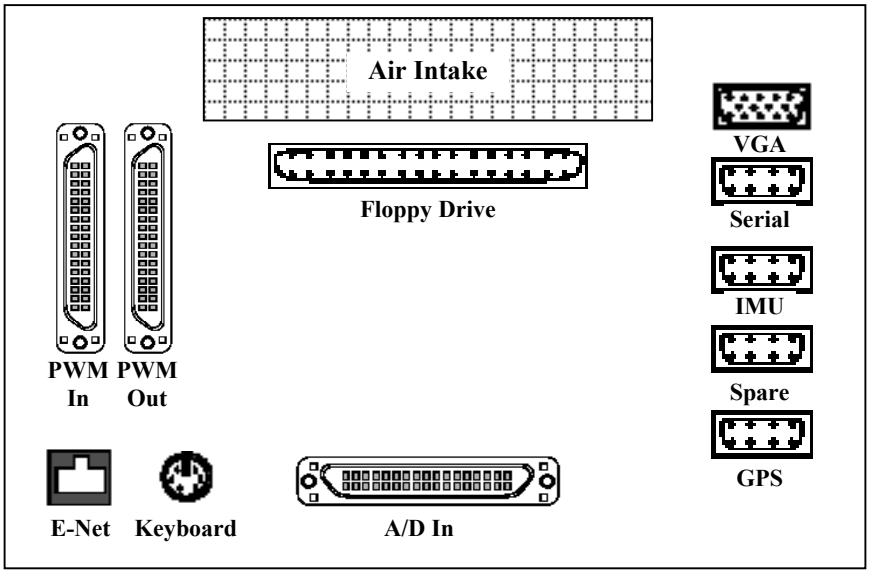


Figure II.5 Top Panel Layout of New Computer

C. SOFTWARE INTERFACE DRIVERS

In order for the airborne computer to compile data from onboard instruments, software interface drivers had to be written. A significant amount of effort was expended in this project to write software interface drivers that will read in and decode data output by the IMU and GPS receiver. This was because existing xPC serial data input block can only read RS-232 messages with fixed sentence structure (i.e. in 32-bit float or 16-bit integer format and each message must end with null character) or binary data stream with fixed number of characters without the capability to identify a header byte or handle varying number of bytes in messages. On the other hand, to work with the IMU and GPS, the interface drivers must be capable of searching for appropriate header byte or header string in a continuous stream of RS-232 data while performing checksum computation in real-time to validate the data and applying appropriate formula to decode the incoming data. Therefore, instrument specific software interface drivers had to be written. Details of the driver implementation for the case of the CrossBow IMU and Ag132 GPS is discussed next.

1. IMU Data Interface

In the Crossbow AHRS400CA-100, the digital data representing each measurement is sent as a 16-bit number (two bytes). The data is sent in 'Big Endien' format, i.e. MSB first then LSB. In Angle Mode, the data generally represents a quantity that can be positive or negative and is sent as a 16-bit signed integer in 2's complement format. Only the timer information and temperature sensor voltage are sent as unsigned integers. Each data packet begins with a header byte (255) and ends with a checksum. Hence, for the Angle Mode used in this project, each message consists of 30 bytes inclusive of the header byte, 14 data values and the checksum as shown in Table II.1.

a. IMU Data Receive Implementation

Several characteristics peculiar to the Crossbow AHRS message data structure shown in Table II.1 dictated the way the software driver receiving serial data from Crossbow had to be written. First, the message rate (i.e. number of message Crossbow outputs) in Continuous Data Collection Mode is not constant. It was measured to be fluctuating between 69.7 Hz and 61.3 Hz in [4]. This precludes the use of a standard message retrieval rate (e.g. fixed at 65 read per second) by the airborne computer if we intend to read and utilize every piece of data output from the instrument. Second, the header byte FF will not be the only FF byte in the each data packet because FF bytes can occur in the message body. Therefore, the receiving software driver must count the bytes received from the serial port and use the checksum to determine if a particular FF byte is a header or just another byte in the body of the message. The implemented Crossbow data receive and decode Simulink block diagram is shown in Figure II.6.

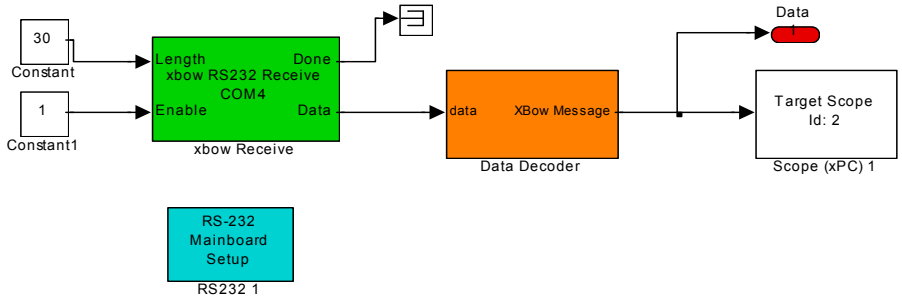


Figure II.6 Crossbow AHRS Data Receive and Decoding Block Diagram

The '*Xbow Receive*' block calls the data receive driver *xbowrcv1p3.c* in Appendix C. *xbowrcv1p3.c* uses a soft-buffer to collect every byte received by the COM port serial buffer and progressively check every byte in the soft-buffer for a FF byte. Upon locating each FF byte (possible message header), the routine computes the checksum of all data bytes (Byte 2 to Byte 29 in Angle Mode) to assess if the computed checksum tallies with the transmitted checksum byte (Byte 30 in Angle Mode case). If the checksum does not tally, that FF byte is not a header and the routines goes in search of the next FF byte. If the checksum tallies, the routine outputs the full message (Header byte + 28 data bytes + checksum byte in Angle Mode case) and shift unused bytes to the front of the soft-buffer for processing at the next time-step. If there is inadequate data to form a message (this occurs if the sampling interval is much smaller than data rate), the previous message is returned during each code execution time-step. If on the other hand, the sampling interval is longer than data rate, and the data in soft-buffer exceeds 100 bytes (i.e. about 3 messages), the soft-buffer is flushed to allow new incoming bytes to reach the front of soft-buffer for processing. This will assure the 'freshness' of data. As such, the implemented driver is able to collect every useful byte of data and operates independently of the message rate from the IMU.

Byte	VG Mode	Scaled Sensor Mode	Voltage Mode
0	Header (255)	Header (255)	Header (255)
1	Roll Angle (MSB)	Roll Angular Rate (MSB)	Roll Gyro Voltage (MSB)
2	Roll Angle (LSB)	Roll Angular Rate (LSB)	Roll Gyro Voltage (LSB)
3	Pitch Angle (MSB)	Pitch Angular Rate (MSB)	Pitch Gyro Voltage (MSB)
4	Pitch Angle (LSB)	Pitch Angular Rate (LSB)	Pitch Gyro Voltage (LSB)
5	Heading Angle (MSB)	Yaw Angular Rate (MSB)	Yaw Gyro Voltage (MSB)
6	Heading Angle (LSB)	Yaw Angular Rate (LSB)	Yaw Gyro Voltage (LSB)
7	Roll Angular Rate (MSB)	X-Axis Acceleration (MSB)	X-Axis Accel Voltage (MSB)
8	Roll Angular Rate (LSB)	X-Axis Acceleration (LSB)	X-Axis Accel Voltage (LSB)
9	Pitch Angular Rate (MSB)	Y-Axis Acceleration (MSB)	Y-Axis Accel Voltage (MSB)
10	Pitch Angular Rate (LSB)	Y-Axis Acceleration (LSB)	Y-Axis Accel Voltage (LSB)
11	Yaw Angular Rate (MSB)	Z-Axis Acceleration (MSB)	Z-Axis Accel Voltage (MSB)
12	Yaw Angular Rate (LSB)	Z-Axis Acceleration (LSB)	Z-Axis Accel Voltage (LSB)
13	X-Ax is Acceleration (MSB)	X-Axis Magnetic Field (MSB)	X-Axis Mag Voltage (MSB)

14	X-Axis Acceleration (LSB)	X-Axis Magnetic Field (LSB)	X-Axis Mag Voltage (LSB)
15	Y-Axis Acceleration (MSB)	Y-Axis Magnetic Field (MSB)	Y-Axis Mag Voltage (MSB)
16	Y-Axis Acceleration (LSB)	Y-Axis Magnetic Field (LSB)	Y-Axis Mag Voltage (LSB)
17	Z-Axis Acceleration (MSB)	Z-Axis Magnetic Field (MSB)	Z-Axis Mag Voltage (MSB)
18	Z-Axis Acceleration (LSB)	Z-Axis Magnetic Field (LSB)	Z-Axis Mag Voltage (LSB)
19	X-Axis Magnetic Field (MSB)	Temp Sensor Voltage (MSB)	Temp Sensor Voltage (MSB)
20	X-Axis Magnetic Field (LSB)	Temp Sensor Voltage (LSB)	Temp Sensor Voltage (LSB)
21	Y-Axis Magnetic Field (MSB)	Time (MSB)	Time (MSB)
22	Y-Axis Magnetic Field (LSB)	Time (LSB)	Time (LSB)
23	Z-Axis Magnetic Field (MSB)	Checksum	Checksum
24	Z-Axis Magnetic Field (LSB)		
25	Temp Sensor Voltage (MSB)		
26	Temp Sensor Voltage (LSB)		
27	Time (MSB)		
28	Time (LSB)		
29	Checksum		

Table II.1 Serial Data Structure From Crossbow AHRS400CA-100

b. Decoding Crossbow Data

The decoding routine simply takes in the float value of consecutive two bytes after the header byte, multiplies the first byte of each pair by 256, adds it to the second byte to obtain the numerical value for data items transmitted in 2's complements. Each set of data (e.g. angles, rates, acceleration and magnetic field) is then scaled according to Simulink implementation structure shown in Figure II.7. The factors AR, GR and MR are specific to each Crossbow AHRS and are given in the factory calibration data of the instrument. An example of the decoded data for Euler angles and rates is presented in Figure II.8 when the Crossbow IMU was rotated in the roll, pitch and yaw axis sequentially by approximately 80° in each direction to check the validity of the decoded data.

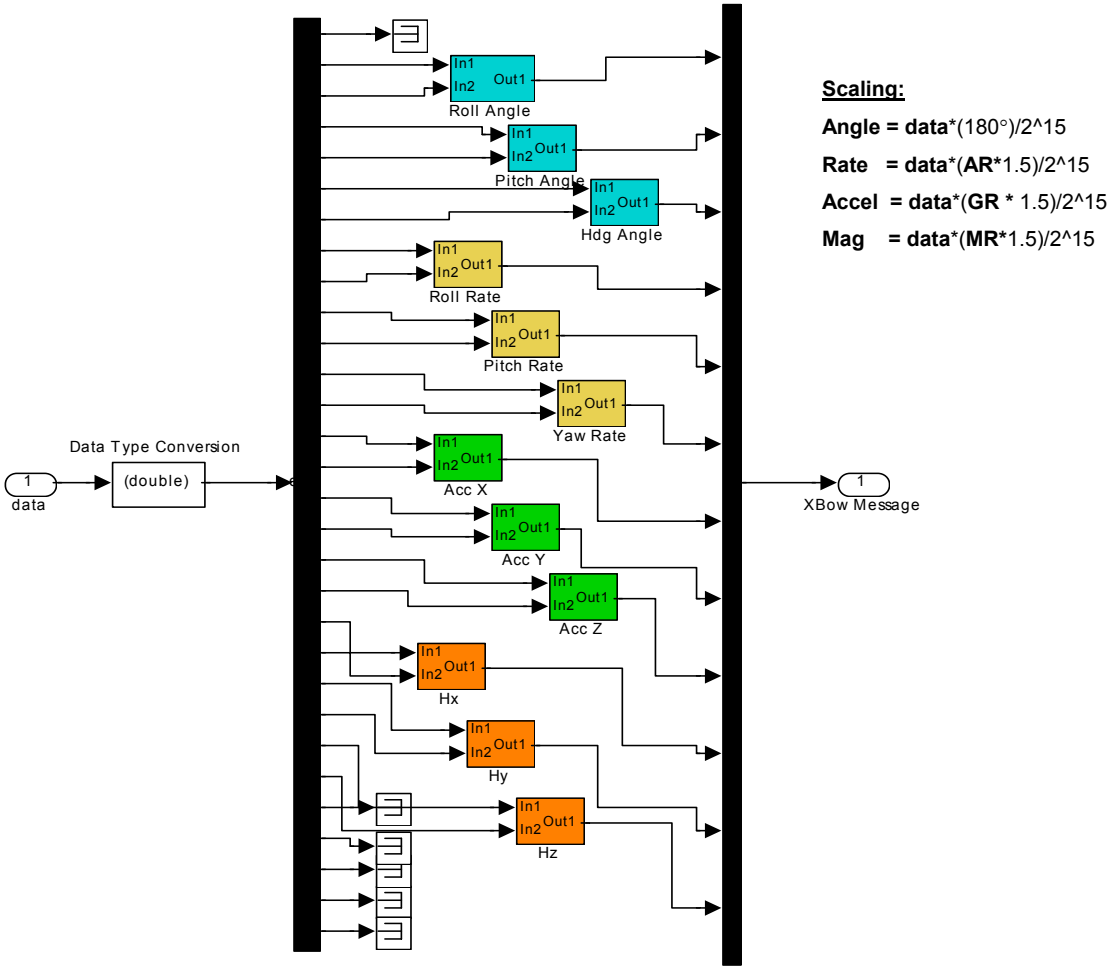


Figure II.7 Simulink Block to Decode Crossbow Data

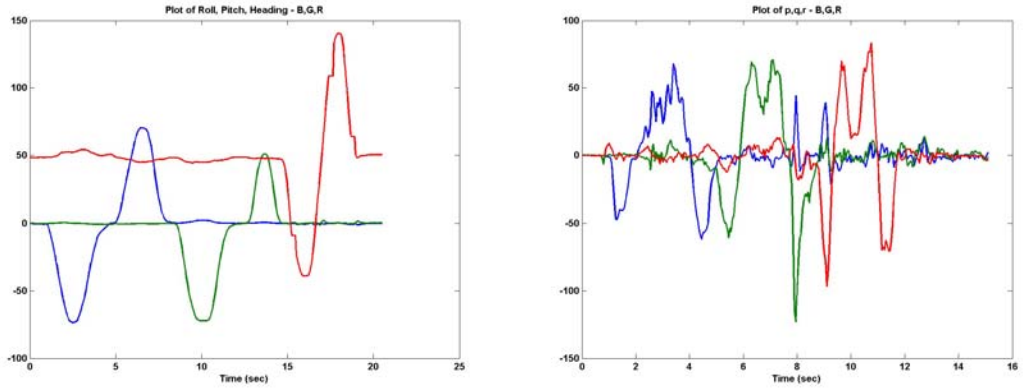


Figure II.8 Decoded Euler Angle and Rates from Crossbow

2. GPS Data Interface

The Ag132 receives GPS messages in NMEA-0183 format. The messages are essentially strings of comma-delimited text shown in Figure II.9 below. Each NMEA message includes a message ID to distinguish the message from other NMEA messages in the data stream. The actual data included in NMEA-0183 messages is placed in fields. Each NMEA message contains different number of fields, and each field is preceded by a comma character. The messages include a checksum value which is useful for checking the integrity of the data included in the message. The checksum is the 8-bit exclusive OR of all the characters in the message, between the '$' and '*' delimiters.

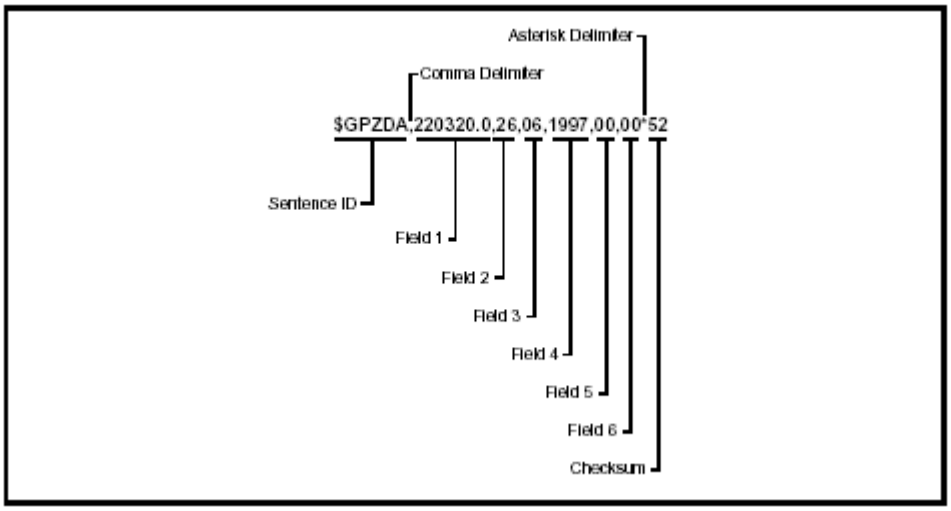


Figure II.9 NMEA-0183 Message Structure

a. *Receiving GPS Message Strings*

The manner in which GPS messages are received the airborne computer is shown in Figure II.10 . The GPS Receive block executes the *gpsrcv.c* software driver in Appendix C to read every byte of the GPS data into a storage and output buffer system (similar to that describe for the Crossbow IMU so that no data byte is lost when sampling is done at data rates higher than the incoming GPS data), evaluates the 5-byte header following each '$' character, identifies if a GPGGA or GPRMC message has been received and returns a Header Index to identify the type of message. It also searches through the message to look for the end of message delimiter bytes (0x0D, 0x0A) which

mark the end of each message and outputs all the bytes between the header string and delimiter bytes.

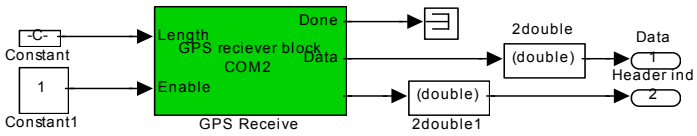


Figure II.10 GPS Message Receive Block Diagram

b. Decoding the GPS Messages

Message decoding is done by two blocks following the *GPS_receive* model as shown in Figure II.11. The *GCA* and *RMC* blocks are enabled to execute the *gpgca.c* and *gprmc.c* routines in Appendix C when the Header Index from the *GPS_receive* model matches the message type each block is assigned to decode. Since each data field is separated by the comma character, the routine extracts each field of useful data from the received binary data sent by the *GPS_receive* model based on the expected format and contents of fields for the GPGGA and GPRMC messages as shown in Table II.2 and Table II.3 respectively.

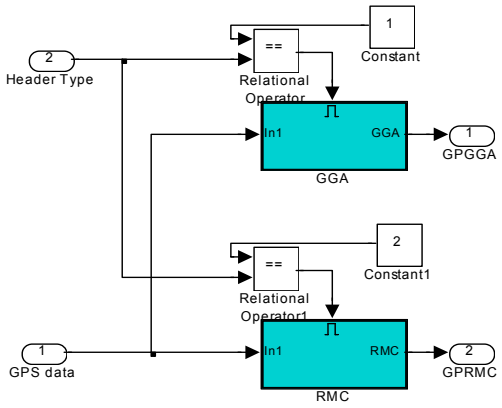


Figure II.11 GPS Message Decoding Block Diagram

Field	Description
1	UTC of position fix in HHMMSS.SS format
2	Latitude in DD MM,MMMM format (0-7 decimal places)

3	Direction of latitude N: North S: South
4	Longitude in DDD MM,MMMM format (0-7 decimal places)
5	Direction of longitude: E: East W: West
6	GPS Quality indicator 0: fix not valid 1: GPS fix 2: DGPS fix
7	Number of SVs in use, 00-12
8	HDOP
9	Antenna height, MSL reference
10	M' indicates that the altitude is in meters.
11	Geoidal separation
12	'M' indicates that the geoidal separation is in meters
13	Age of differential GPS data record, Type 1. Null when DGPS not used
14	Base station ID, 0000-1023

Table II.2 GGA Message Fields

Field	Description
1	Time: UTC time of the position fix in hhmmss.ss format
2	Status A: Valid V: Navigation Receiver Warning (V is output whenever the Receiver suspects something is wrong)
3	Latitude coordinate (the number of decimal places, 0–7, is programmable and determined by the numeric precision selected in TSIP Talker for a RMC message)
4	Latitude direction: N = North, S = South
5	Longitude coordinate (the number of decimal places, 0–7, is programmable and determined by the numeric precision selected in TSIP Talker for a RMC message)
6	Longitude direction W: West E: East
7	Speed Over Ground (SOG) in knots (0–3 decimal places)
8	Track Made Good, True, in degrees
9	Date in dd/mm/yy format
10	Magnetic Variation in degrees
11	Direction of magnetic variation E: Easterly variation from True course (subtracts from True course) W: Westerly variation from True course (adds to True course)
12	Mode Indication A: Autonomous D: Differential N: Data not valid

Table II.3 RMC Message Fields

D. COMBINED I/O TEST

The Simulink block diagram for the combined I/O test involving GPS data, IMU data, PWM generation, PWM capture and host-target communication via serial modem is shown in Figure II.12. It is driven by user inputs from the display screen in Figure II.13 created using Dials & Gauges. The PWM signal generated by the computer (shown in Figure II.14) was designed to emulate that generated by the Futaba® remote control system which uses a pulse period of approximately 14.25 milliseconds and a pulse width varying from around 0.9 to 2.2 milliseconds depending on the command input.

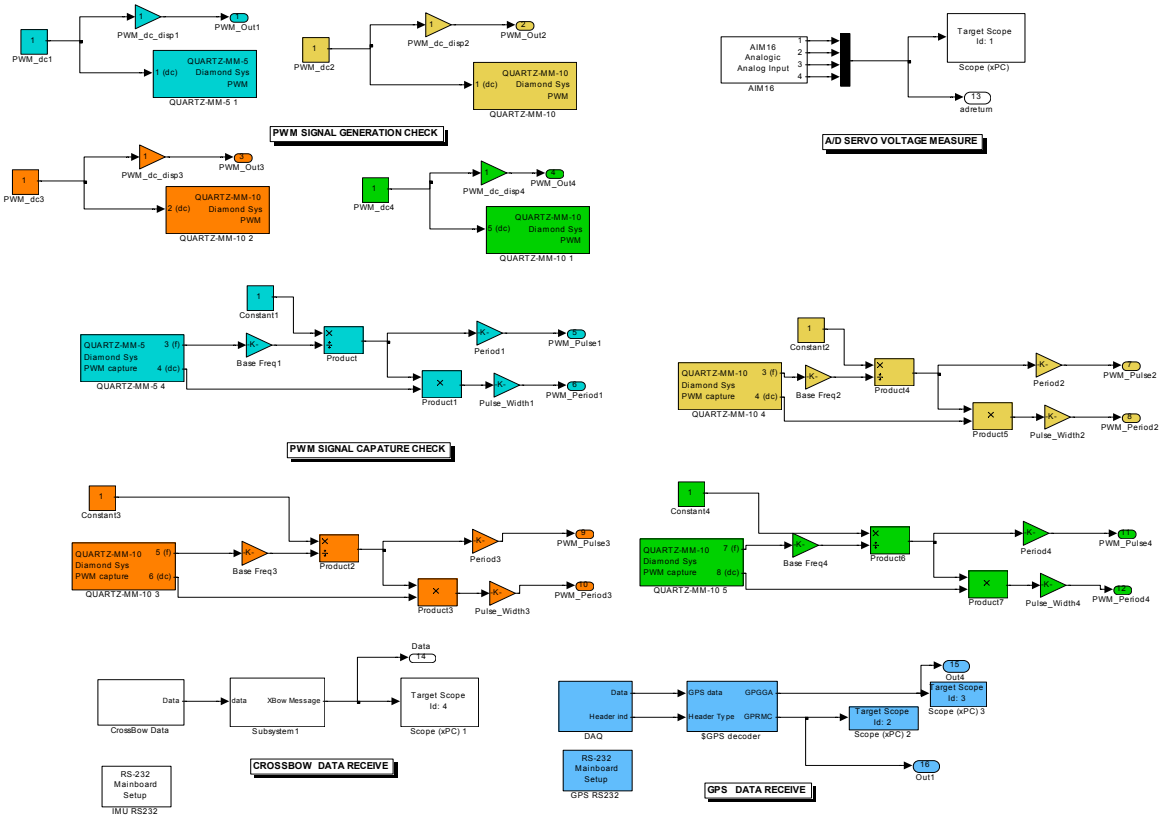


Figure II.12 Block Diagram For Combined Test

The computer takes only an average of 350 microsecond to compile the GPS, IMU, A/D sensor data, output PWM signals, measure PWM signals and display results to the VGA monitor at each time step. This execution time can further be reduced if display

of data on the xPC scope is removed. Hence, the data collection and processing time on the airborne computer has improved significantly over the original setup where data was updated only at 25 Hz.

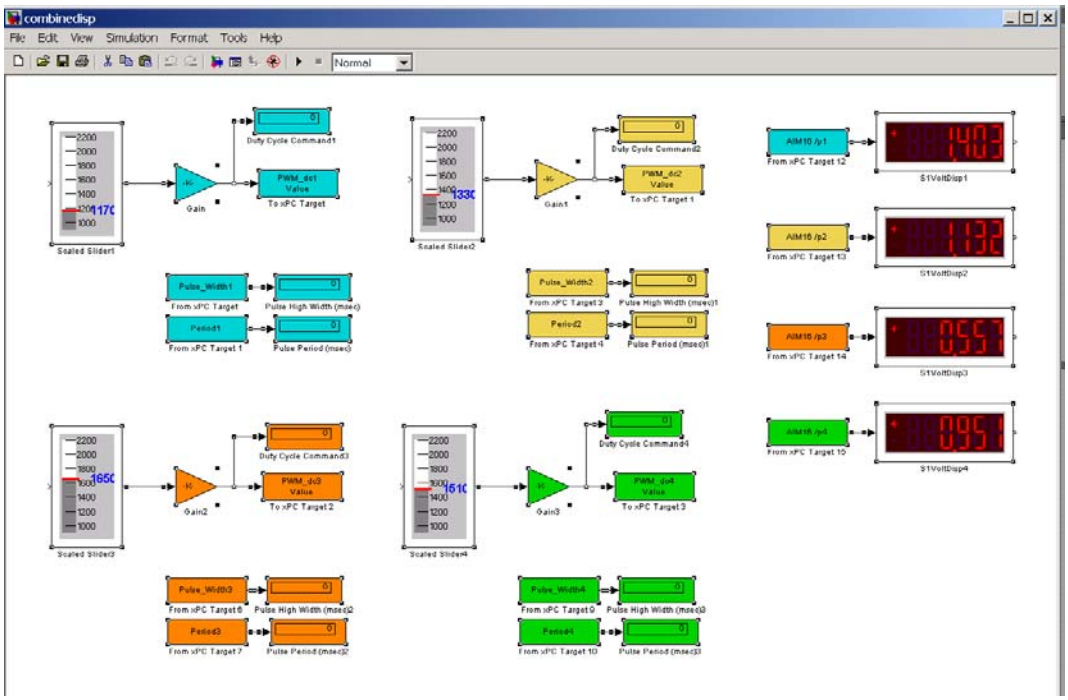


Figure II.13 Combined Test User Interface Screen

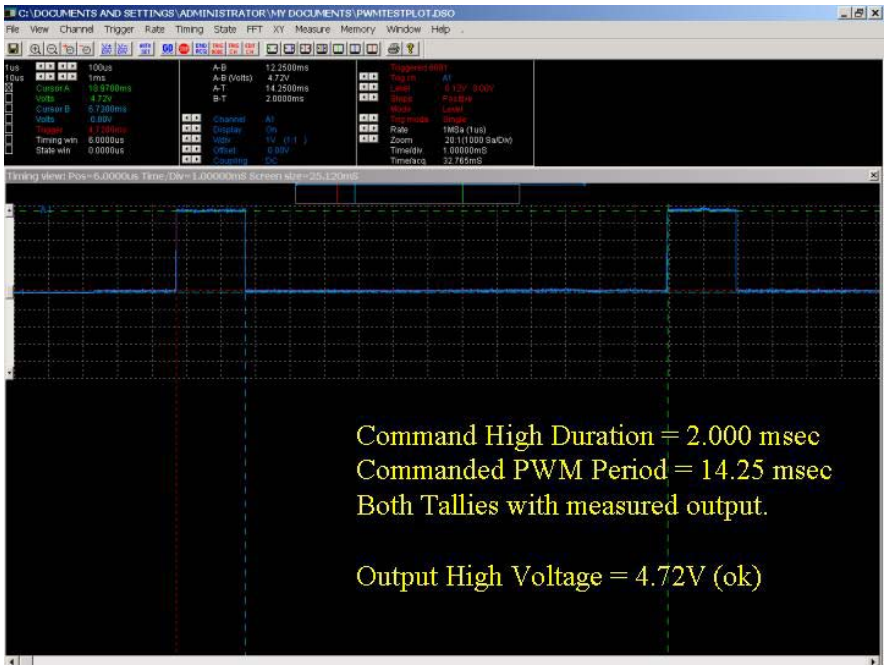


Figure II.14 PWM Signal Generated by Computer Matches Command

III. FLIGHT CONTROLLER DESIGN

For the controller design effort, a six degrees-of-freedom (6-DOF) model of the NPS FROG UAV was first developed in Simulink based on available flight characteristics data collected in [6]. The flight controller was then designed using classical control inner-outer-loop approach using the dynamic model of the FROG UAV around the trimmed flight condition and adjusted for non-linearity with gain-scheduling using dynamic pressure as the scheduling parameter. The inner-loop-outer-loop structure was chosen as the first controller to be implemented because its structure made it easier to conduct checks on the aircraft. Another controller was designed using the Integral Linear Quadratic Regulator (Linear LQR) Synthesis method described in [8] to serve as performance comparison.

A. 6DOF AIRCRAFT MODEL DEVELOPMENT

The derivation of equations of motion for a 6-DOF aircraft model can be divided into 2 main parts. The first part is the formulation of equations of motion for the aircraft, treated as a rigid body, in space. These dynamics are typically common for any rigid body. The second part is the computation of aerodynamic, gravitational and thrust forces on the aircraft. These forces, except for gravitational, are specific to each aircraft and depend directly on its stability and control derivatives as well as on engine performance.

1. Equations of Motions

The equations of motion for the linear dynamics and angular dynamics of the aircraft is developed as follows:

a. *Linear Dynamics Equations*

The equations for linear motion are governed by Newton's Second Law, $F = m\,a$, expressed in the inertial frame $\{U\}$. However, aircraft velocities and attitude angles are usually measured with respect to the aircraft's body axis coordinate system $\{B\}$. Linear forces and accelerations are also typically expressed in $\{B\}$. The origin of

the body axis (*BO*) is taken to coincide with the aircraft center of gravity (c.g.), hence the position of aircraft c.g. with respect to inertial axis is denoted ${}^{U}P_{BO}$ and its velocity is given by

$$ {}^{U}v_{BO} \triangleq {}^{U}\dot{P}_{BO} \tag{III.1} $$

Pre-multiplying by the rotation matrix from $\{U\}$ to $\{B\}$ gives

$$ {}^{B}_{U}R \, {}^{U}v_{BO} \triangleq {}^{B}_{U}R \, {}^{U}\dot{P}_{BO} \tag{III.2} $$

where the rotation matrix ${}^{B}_{U}R$ is given by

$$ {}^{B}_{U}R = \begin{bmatrix} \cos\Psi\cos\Theta & \sin\Psi\cos\Theta & -\sin\Theta \\ \cos\Psi\sin\Theta\sin\Phi - \sin\Psi\cos\Phi & \sin\Psi\sin\Theta\sin\Phi + \cos\Psi\cos\Phi & \cos\Theta\sin\Phi \\ \cos\Psi\sin\Theta\cos\Phi + \sin\Psi\sin\Phi & \sin\Psi\sin\Theta\cos\Phi - \cos\Psi\sin\Phi & \cos\Theta\cos\Phi \end{bmatrix} \tag{III.3} $$

or equivalently as

$$ {}^{B}v_{BO} = {}^{B}\dot{P}_{BO} \tag{III.4} $$

The total derivative of a vector A in a rotating coordinate system with angular velocity ω, is given by Coriolis' theorem

$$ \dot{A} = \frac{d}{dt}A + \omega \times A \tag{III.5} $$

Hence, the rigid body's linear acceleration is given by

$$ {}^{B}\dot{v}_{BO} = \frac{d}{dt}{}^{B}v_{BO} + {}^{B}\omega_{B} \times {}^{B}v_{BO} \tag{III.6} $$

Newton's 2nd law applied to the aircraft c.g. can then written as (III.7) resolved in inertial frame or as (III.8) resolved in body axes.

$$ \begin{aligned} {}^{U}F &= m \, {}^{U}a_{BO} \\ &= m \, {}^{U}\dot{v}_{BO} \end{aligned} \tag{III.7} $$

$$ \begin{aligned} {}^{B}F &= m \, {}^{B}_{u}R \, {}^{u}\dot{v}_{BO} \\ &= m \, {}^{B}\dot{v}_{BO} \end{aligned} \tag{III.8} $$

Finally, the equation to be modeled is given by (III.9) after incorporating (III.6) into (III.8).

$$ {}^{B}F = m\left(\frac{d}{dt} {}^{B}v_{BO} + {}^{B}\omega_{B} \times {}^{B}v_{BO} \right) \tag{III.9} $$

b. Angular Dynamics Equations

The angular equations of motion are derived using Euler's law for conservation of angular momentum in the inertial frame where ${}^{U}N_{BO}$ denotes moment imparted to the rigid body and ${}^{U}\dot{L}_{BO}$ denotes the rate of change of angular momentum.

$$ {}^{u}N_{BO} = {}^{u}\dot{L}_{BO} \tag{III.10} $$

Again using the rotation matrix to body axes and applying Coriolis' Theorem

$$ {}^{B}\dot{L}_{BO} = {}^{B}_{u}R \, {}^{u}N_{BO} \tag{III.11} $$

$$ {}^{B}\dot{L}_{BO} = \frac{d}{dt} {}^{B}L_{BO} + {}^{B}\omega_{B} \times {}^{B}L_{BO} \tag{III.12} $$

where the angular momentum term ${}^{B}L_{BO}$ is defined as the product of the inertia tensor and the body's angular velocity plus the inertia tensor of the spinning rotor on the aircraft and the rotor's angular velocity all with respect to body axes.

$$ {}^{B}L_{BO} \triangleq I_{B} \, {}^{B}\omega_{B} + I_{R} \, {}^{B}\omega_{R} \tag{III.13} $$

Substituting (III.13) into (III.12) gives

$$ {}^{B}\dot{L}_{BO} = \frac{d}{dt}\left(I_{B} \, {}^{B}\omega_{B} + I_{R} \, {}^{B}\omega_{R} \right) + {}^{B}\omega_{B} \times \left(I_{B} \, {}^{B}\omega_{B} + I_{R} \, {}^{B}\omega_{R} \right) \tag{III.14} $$

In this project, $I_{R} \, {}^{B}\omega_{R}$ is also neglected. If we further assume that I_{B}, I_{R} and ${}^{B}\omega_{R}$ are constant, (III.15) results.

$$ {}^{B}N_{BO} = I_{B} \, {}^{B}\dot{\omega}_{B} + {}^{B}\omega_{B} \times \left(I_{B} \, {}^{B}\omega_{B} + I_{R} \, {}^{B}\omega_{R} \right) \tag{III.15} $$

c. State Equations

In summary, the kinematics equations to be implemented in the rigid body 6DOF model is governed by (III.16) and is re-written in matrix form as (III.17).

$$\begin{bmatrix} {}^{B}F \\ {}^{B}N \end{bmatrix} = \begin{bmatrix} m\dfrac{d}{dt}{}^{B}v_{BO} + m\left({}^{B}\omega_{B} \times {}^{B}v_{BO}\right) \\ I_{B}\,{}^{B}\dot{\omega}_{B} + {}^{B}\omega_{B} \times \left(I_{B}\,{}^{B}\omega_{B} + I_{R}\,{}^{B}\omega_{B}\right) \end{bmatrix} \tag{III.16}$$

which can be re-arranged and implemented in linear state space form as

$$\frac{d}{dt}\begin{bmatrix} {}^{B}v_{BO} \\ {}^{B}\omega_{B} \end{bmatrix} = \begin{bmatrix} -{}^{B}\omega_{B} \times {}^{B}v_{BO} + \dfrac{{}^{B}F}{m} \\ -I_{B}^{-1}\,{}^{B}\omega_{B} \times \left(I_{B}\,{}^{B}\omega_{B} + I_{R}\,{}^{B}\omega_{R}\right) + I_{B}^{-1}\,{}^{B}N \end{bmatrix} \tag{III.17}$$

2. Forces and Moments on Aircraft

In order to compute the ${}^{B}F$ and ${}^{B}N$ as required in (III.17), the total contribution of forces and moments exerted on the aircraft due to aerodynamic, propulsion and gravitational effects is computed using the expression (III.18).

$$\begin{bmatrix} {}^{B}F \\ {}^{B}N \end{bmatrix} = \begin{bmatrix} {}^{B}F_{Aero} + {}^{B}F_{\mathrm{Pr}op} + {}^{B}F_{\mathrm{Pr}op} \\ {}^{B}N_{Aero} + {}^{B}N_{\mathrm{Pr}op} \end{bmatrix} \tag{III.18}$$

a. Aerodynamic Forces and Moments

The aerodynamics force and moment terms are determined using a first-order Taylor series expansion around the aircraft trimmed operating point. Each term in the series is a partial derivative of ${}^{B}F$ and ${}^{B}N$ with respect to the aerodynamic variables $\frac{u}{V_T}, \alpha, \beta, p, q, r$ i.e.

$$F_{Aero} = \delta F_{x'}x' + \delta F_{\dot{x}'}\dot{x}' + \delta F_{\Delta}\Delta + F_{0} \tag{III.19}$$

$$N_{Aero} = \delta N_{x'}x' + \delta N_{\dot{x}'}\dot{x}' + \delta N_{\Delta}\Delta + N_{0}$$

where

$$x' = \left[\frac{u}{V_T}, \beta, \alpha, \frac{pb}{2V_T}, \frac{qc}{2V_T}, \frac{rb}{2V_T}\right] \quad \text{(III.20)}$$

$$\dot{x}' = \left[\dot{\beta}, \dot{\alpha}\right] \quad \text{(III.21)}$$

$$\Delta = \left[\delta_e, \delta_r, \delta_a\right] \quad \text{(III.22)}$$

The forces and moments generated by aerodynamic forces with respect to the wind-axis as denoted in (III.23) and the rotation matrix ${}^{B}_{w}R$ defined in (III.24) can be applied to obtained ${}^{B}F_A$ and ${}^{B}N_A$.

$$\begin{bmatrix} {}^{w}F_A \\ {}^{w}N_A \end{bmatrix} = \bar{q}\,\bar{S}\left\{C_{F_0} + \frac{\partial C}{\partial x'}x' + \frac{\partial C}{\partial \dot{x}'}\dot{x}' + \frac{\partial C}{\partial \Delta}\Delta\right\} \quad \text{(III.23)}$$

$${}^{B}_{w}R = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta & -\sin\alpha \\ -\sin\beta & \cos\beta & 0 \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta & \cos\alpha \end{bmatrix} \quad \text{(III.24)}$$

The forces resulting from drag and lift are taken negative to obtain positive signed forces in the x_B (i.e. forward) and z_B (i.e. down) directions.

$${}^{w}F_{Aero} = \begin{bmatrix} -D \\ Y \\ -L \end{bmatrix} \quad \text{and} \quad {}^{w}N_{Aero} = \begin{bmatrix} l \\ m \\ n \end{bmatrix} \quad \text{(III.25)}$$

The rest of the quantities in (III.23) are defined as follows:

$$\frac{\partial C}{\partial x'} = \begin{bmatrix} C_{D_U} & C_{D_\beta} & C_{D_\alpha} & C_{D_p} & C_{D_q} & C_{D_r} \\ C_{Y_U} & C_{Y_\beta} & C_{Y_\alpha} & C_{Y_p} & C_{Y_q} & C_{Y_r} \\ C_{L_U} & C_{L_\beta} & C_{L_\alpha} & C_{L_p} & C_{L_q} & C_{L_r} \\ C_{l_U} & C_{l_\beta} & C_{l_\alpha} & C_{l_p} & C_{l_q} & C_{l_r} \\ C_{m_U} & C_{m_\beta} & C_{m_\alpha} & C_{m_p} & C_{m_q} & C_{m_r} \\ C_{n_U} & C_{n_\beta} & C_{n_\alpha} & C_{n_p} & C_{n_q} & C_{n_r} \end{bmatrix} \quad \text{(III.26)}$$

$$\frac{\partial C}{\partial \dot{x}'} = \begin{bmatrix} 0 & C_{D_{\dot{\beta}}} & C_{D_{\dot{\alpha}}} & 0 & 0 & 0 \\ 0 & C_{Y_{\dot{\beta}}} & C_{Y_{\dot{\alpha}}} & 0 & 0 & 0 \\ 0 & C_{L_{\dot{\beta}}} & C_{L_{\dot{\alpha}}} & 0 & 0 & 0 \\ 0 & C_{l_{\dot{\beta}}} & C_{l_{\dot{\alpha}}} & 0 & 0 & 0 \\ 0 & C_{m_{\dot{\beta}}} & C_{m_{\dot{\alpha}}} & 0 & 0 & 0 \\ 0 & C_{n_{\dot{\beta}}} & C_{n_{\dot{\alpha}}} & 0 & 0 & 0 \end{bmatrix} \tag{III.27}$$

$$\frac{\partial C}{\partial \Delta} = \begin{bmatrix} C_{D_{\delta e}} & C_{D_{\delta r}} & C_{D_{\delta a}} & C_{D_{\delta T}} \\ C_{Y_{\delta e}} & C_{Y_{\delta r}} & C_{Y_{\delta a}} & C_{Y_{\delta T}} \\ C_{L_{\delta e}} & C_{L_{\delta r}} & C_{L_{\delta a}} & C_{L_{\delta T}} \\ C_{l_{\delta e}} & C_{l_{\delta r}} & C_{l_{\delta a}} & C_{l_{\delta T}} \\ C_{m_{\delta e}} & C_{m_{\delta r}} & C_{m_{\delta a}} & C_{m_{\delta T}} \\ C_{n_{\delta e}} & C_{n_{\delta r}} & C_{n_{\delta a}} & C_{n_{\delta T}} \end{bmatrix} \tag{III.28}$$

$$C_{F_0} = \begin{bmatrix} C_{D_0} \\ C_{Y_0} \\ C_{L_0} \\ C_{l_0} \\ C_{m_0} \\ C_{n_0} \end{bmatrix} \tag{III.29}$$

$$q = \frac{1}{2} \rho V_T^{\,2} \tag{III.30}$$

$$\overline{S} = \begin{bmatrix} -S & 0 & 0 & 0 & 0 & 0 \\ 0 & S & 0 & 0 & 0 & 0 \\ 0 & 0 & -S & 0 & 0 & 0 \\ 0 & 0 & 0 & Sb & 0 & 0 \\ 0 & 0 & 0 & 0 & Sc & 0 \\ 0 & 0 & 0 & 0 & 0 & Sb \end{bmatrix} \tag{III.31}$$

The aerodynamic variables x' are typically not used as states in the computation of ${}^{B}F_{A}$ and ${}^{B}N_{A}$. Instead, the set of orthogonal linear velocities and angular velocities in (III.32) which are easily measured on the aircraft is commonly used.

$$x = \begin{bmatrix} u & v & w & p & q & r \end{bmatrix}^{T} \tag{III.32}$$

Transformation matrices M' and $\dot{M}'$ relating the chosen state vector x to the aerodynamic variables x' and $\dot{x}'$ are defined as follows:

$$M' : x \rightarrow x'$$
$$\dot{M}' : \dot{x} \rightarrow \dot{x}'$$

where

$$M' = \begin{bmatrix} 1/V_T & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/V_T & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/V_T & 0 & 0 & 0 \\ 0 & 0 & 0 & b/2V_T & 0 & 0 \\ 0 & 0 & 0 & 0 & c/2V_T & 0 \\ 0 & 0 & 0 & 0 & 0 & b/2V_T \end{bmatrix} \tag{III.33}$$

$$\dot{M}' = \begin{bmatrix} 0 & c/2V_T^{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & b/2V_T^{2} & 0 & 0 & \end{bmatrix} \tag{III.34}$$

The complete expression for ${}^{B}F_{A}$ and ${}^{B}N_{A}$ can then be written as

$$\begin{bmatrix} {}^{B}F_{A} \\ {}^{B}N_{A} \end{bmatrix} = \begin{bmatrix} {}^{B}_{w}R & 0 \\ 0 & {}^{B}_{w}R \end{bmatrix} \bar{q}\,\bar{S} \left\{ C_{F_0} + \frac{\partial C}{\partial x'} M' x + \frac{\partial C}{\partial \dot{x}'} \dot{M}' \dot{x} + \frac{\partial C}{\partial \Delta} \Delta \right\} \tag{III.35}$$

b. Gravitational and Propulsive Forces and Moments

Gravity contributes forces on the rigid body but no moments since the forces are assumed to act at the center of gravity. The gravitational forces acting on the aircraft are obtained simply by pre-multiplying ${}^{U}F_{Grav}$ by the appropriate rotation matrix as follows:

$$ {}^{U}F_{Grav} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{III.36} $$

then

$$ {}^{B}F_{Grav} = {}^{B}_{U}R \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{III.37} $$

Propulsive forces and moments are exerted in $\{B\}$ axis and is simply computed directly based on the expressions

$$ {}^{B}F_{\mathrm{Pr}op} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \tag{III.38} $$

$$ {}^{B}N_{\mathrm{Pr}op} = \begin{bmatrix} T_l \\ T_m \\ T_n \end{bmatrix} \tag{III.39} $$

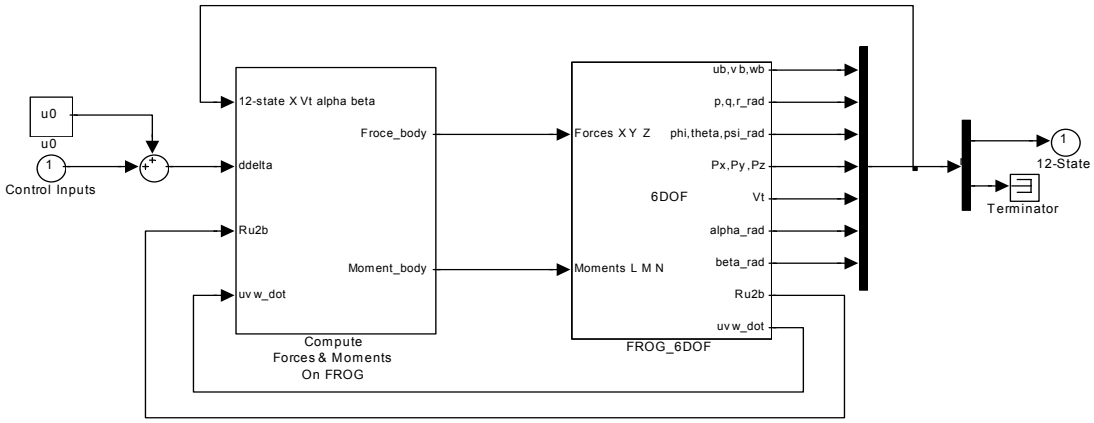
where T_i's represents forces or moments due to propulsive thrust. Obviously, the propulsive forces and moments depend on the engine installation and must be determined for each aircraft individually. For the purpose of this project, the thrust forces in ${}^{B}y$, ${}^{B}z$ and moments are considered negligible and are ignored.

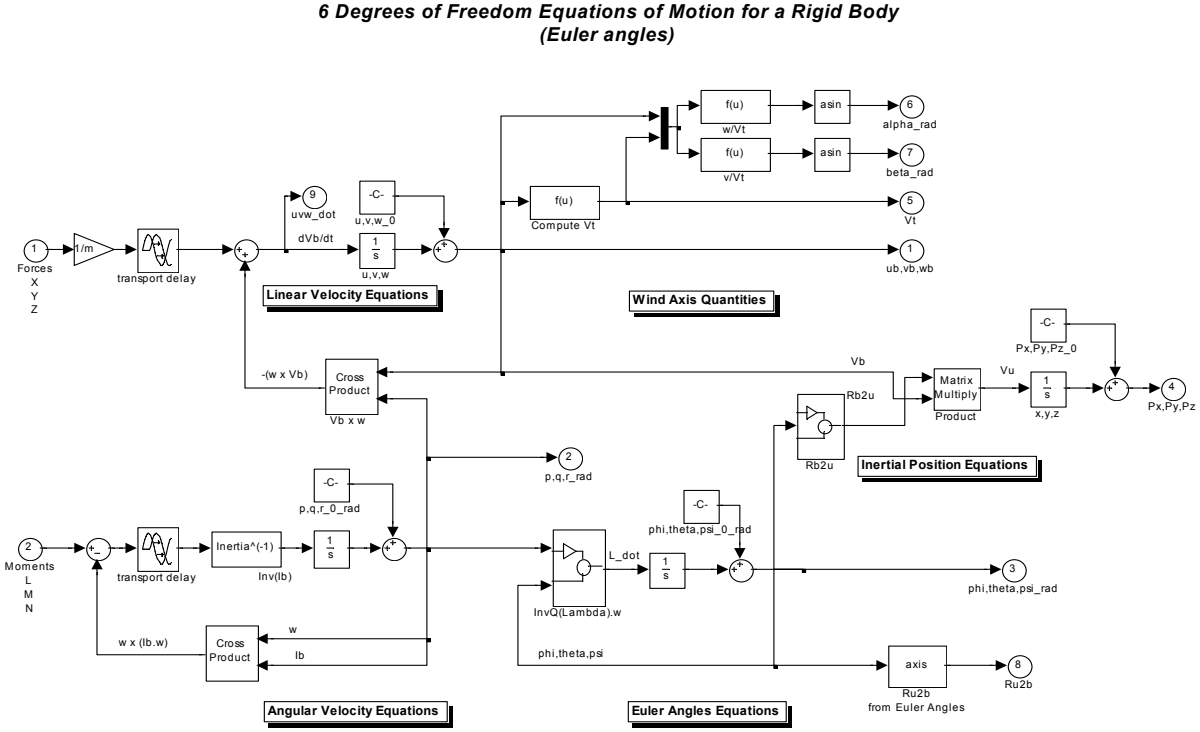
The complete forces and moments exerted on the aircraft are thus given by (III.40). It is implemented in Simulink using block diagrams shown in Figure III.1 and forms the input to the Equations of Motions portion of the 6DOF model.

$$ \begin{bmatrix} {}^{B}F \\ {}^{B}N \end{bmatrix} = \left\{ \begin{bmatrix} {}^{B}_{w}R & 0 \\ 0 & {}^{B}_{w}R \end{bmatrix} \bar{q}\,\bar{S} \left\{ C_{F_0} + \frac{\partial C}{\partial x'} M' x + \frac{\partial C}{\partial \dot{x}'} \dot{M}' \dot{x} + \frac{\partial C}{\partial \Delta} \Delta \right\} + \begin{bmatrix} {}^{B}F_{Grav} \\ 0 \end{bmatrix} + \begin{bmatrix} {}^{B}F_{\mathrm{Pr}op} \\ 0 \end{bmatrix} \right\} \tag{III.40} $$

3. 6DOF Model of FROG UAV in Simulink

The Simulink models implementing Equations (III.17) and (III.40) are shown in the following figures. This non-linear 6DOF model of the FROG UAV was then used to design controllers for the aircraft.





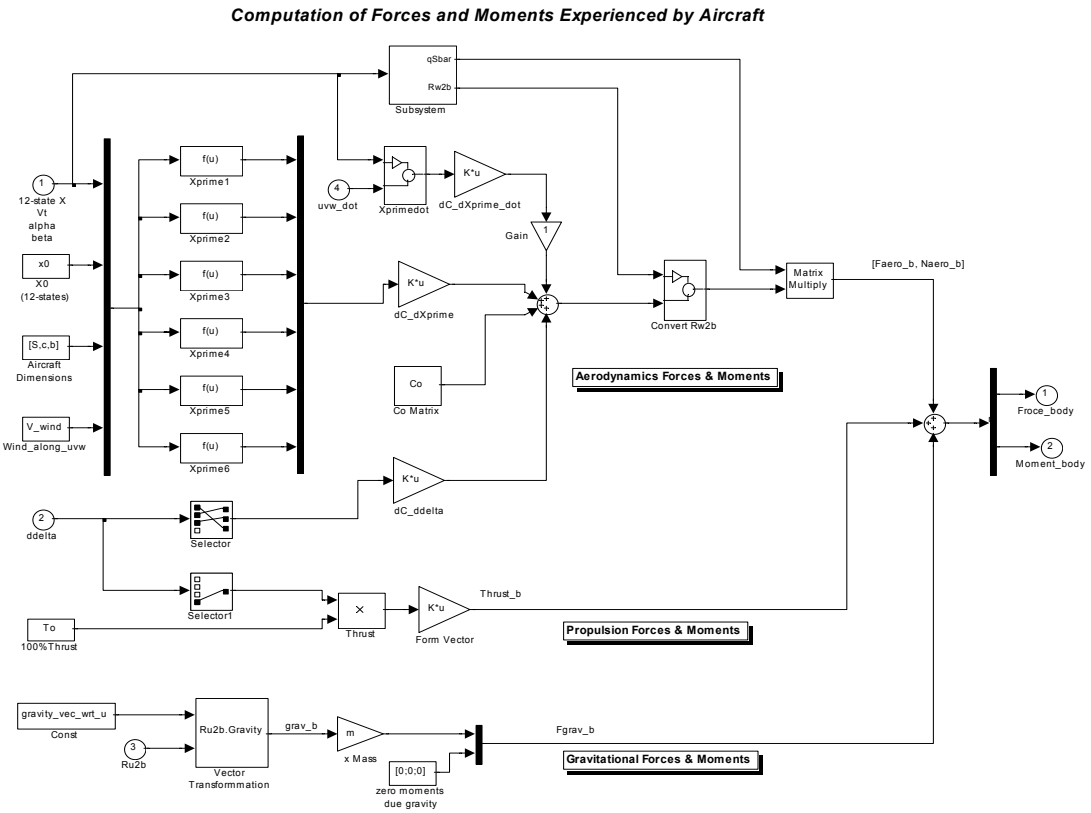


Figure III.1 Simulink Blocks Implementing 6DOF Model of FROG UAV

B. CLASSICAL CONTROLLER DESIGN

The first autopilot was designed based on classical inner/outer loop methodology using the linearized model of the 6DOF FROG UAV developed in the previous section. Actuator models representative of those driving surface actuators and throttle on the aircraft with saturation limits imposed were included before the non-linear FROG model to better emulate the actual hardware response. The procedure consists of evaluating the stability of the feedback loop using root locus techniques, adding poles or zeros to shape the system behavior in the compensator where needed, adjusting loop gains to achieve desired gain and phase margins and verifying the response in each loop with step commands of reasonable magnitude.

General requirements were to attain more than 6dB gain margin, at least 45° phase margin and at least one decade bandwidth separation between inner and outer

loops. The control channels were designed in the order: yaw damper, speed control, altitude control followed by heading control

1. Yaw Damper Design

The yaw damper was designed first in order to prevent the aircraft from yawing due to its lightly damped dutch-roll mode when it executes a longitudinal maneuver. This loop was comparatively more difficult than the rest due to the presence of low damping zeros near the origin and a unstable pole from the spiral mode. As shown in Figure III.2, the root locus for the yaw damper near the origin would result in instability or very poor damping due to the zeros at -0.0834±0.8298i with a simple gain feedback. Poles had to be added at -0.5 and -0.3, with zeros at -0.5000±1.0000i to bring the locus quickly into the stable region and give good damping as shown in the second root locus in Figure III.2.

For turn coordination, a computed rate bias of $g\tan\phi_c / V_T$ was imposed as a command to the PI controller in the yaw damper as described in [7] so that yaw damper would not attempt to counter a commanded turn. The final structure of the yaw damper is shown in Figure III.3. The airplane response responses to yaw rate disturbance inject of 0.2 rad/s and commanded yaw rate of 0.2 rad/s is presented in Figure III.4. The yaw rate feedback method does not give extremely good responses due to the influence of the unstable pole from the spiral mode.

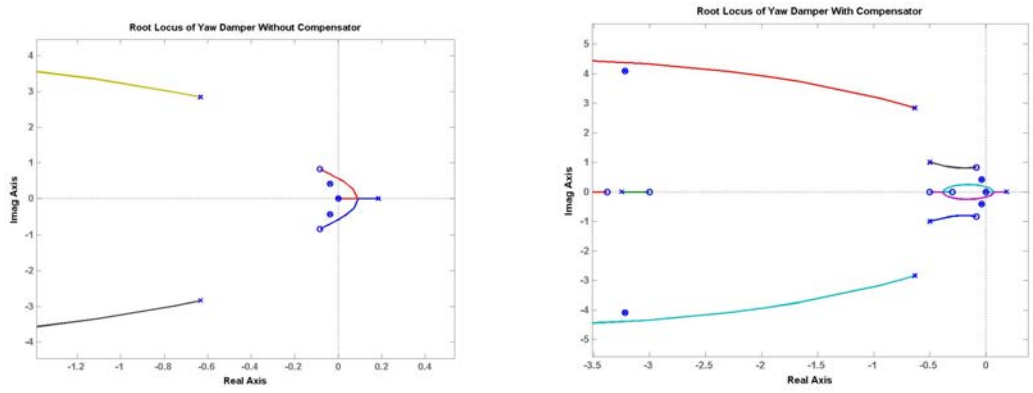


Figure III.2 Root Locus of Yaw Damper With and Without Compensator

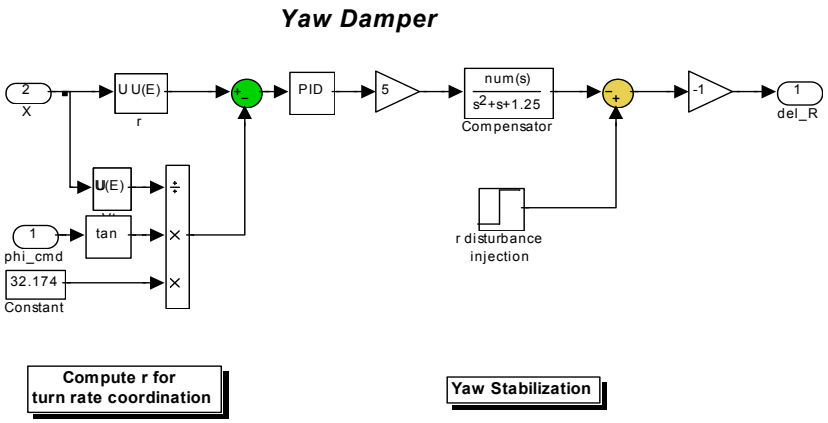


Figure III.3 Yaw Damper Block Diagram

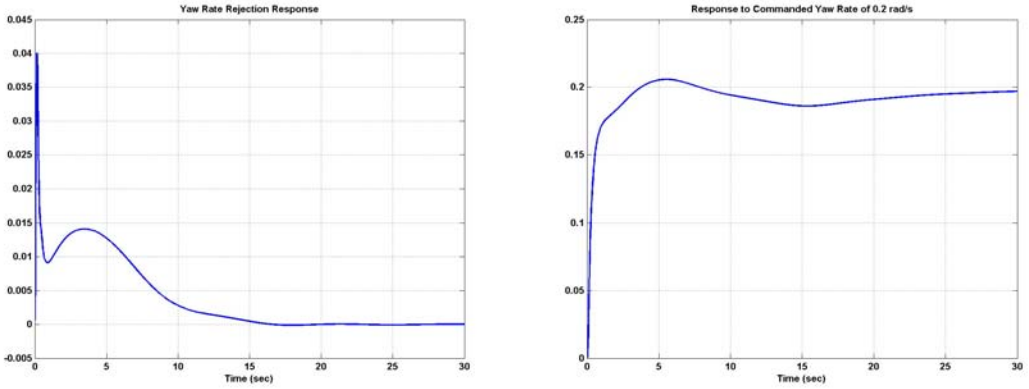


Figure III.4 Yaw Damper Responses

2. Speed Controller

A simple PI controller was implemented for speed control as shown in Figure III.5. This was enough to give a phase margin 91.8° @ 0.42 rad/s with infinite gain margin. Figure III.6 shows that the aircraft would be able to track a commanded 10 fps change in airspeed in about 10 seconds.

Speed Controller

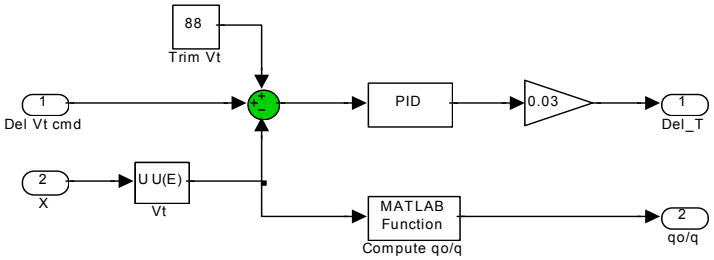


Figure III.5 Speed Control Block Diagram

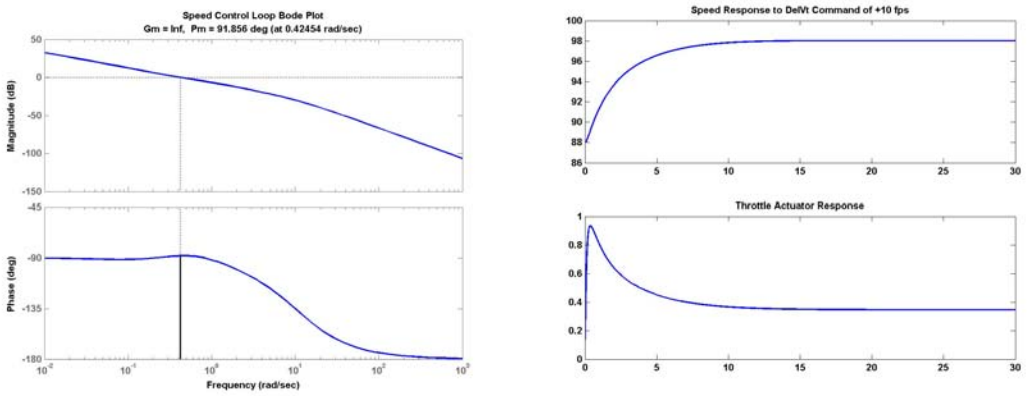


Figure III.6 Speed Controller Responses

3. Altitude Controller

Altitude control entails driving the elevator of the aircraft to pitch the aircraft up or down in a climb or dive maneuver to execute desired altitude changes. To achieve this, the aircraft must first be capable of pitch control. Hence, the pitch loop is often referred to as the inner loop and is designed first. The pitch control loop consists of a PID compensator on the pitch attitude error (θ_e) while the outer loop employed another PID compensator with altitude error, which generates the pitch command for the inner loop. Additionally, a compensator with pole at –25 and zero at –15 had to be inserted to delay the system poles from crossing into the right half plane. This was because the longitudinal modes originally had a real zero at s=19.4 and would result in an odd number of poles and zeros to the right of the origin and the integrator pole from the outer

loop PID would go right immediately otherwise. The resulting controller structure and its root locus is presented in Figure III.7 and Figure III.8 respectively.

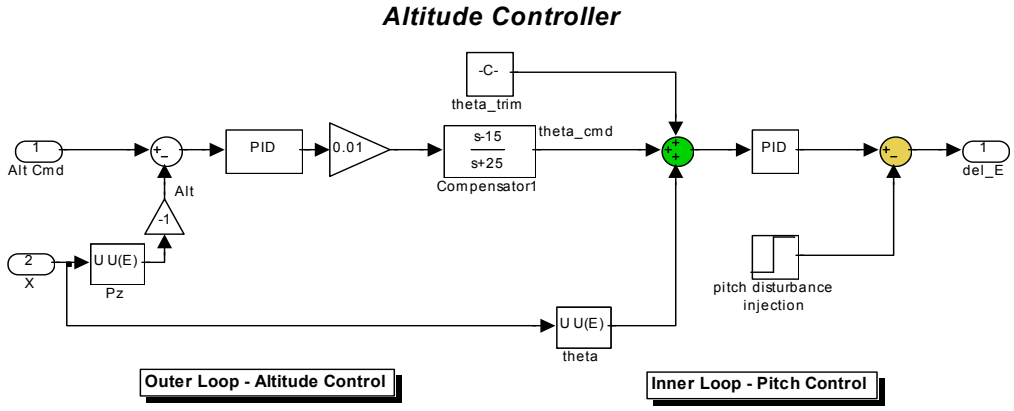


Figure III.7 Altitude Controller Block Diagram

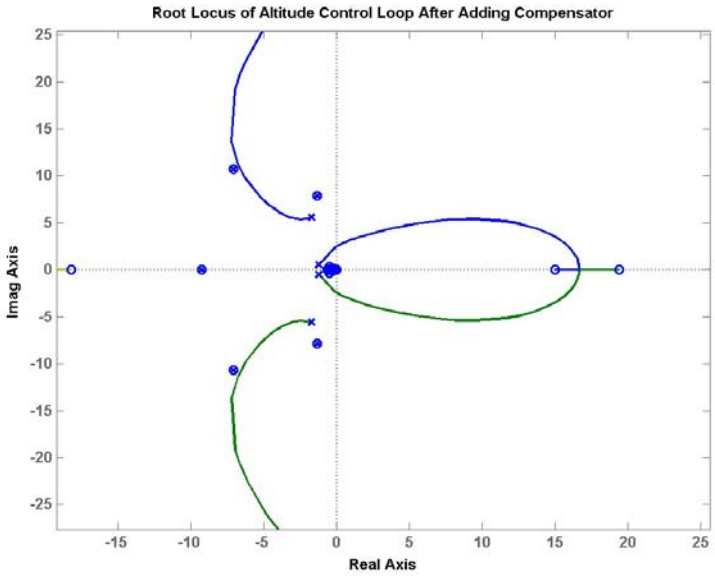


Figure III.8 Root Locus of Altitude Controller With Compensator

Figure III.9 shows the pitch control loop responses to commands from the outer loop and its disturbance rejection property. In both cases, when 0.2 rad changes were input, the elevator deflected a reasonable 0.1 rad or 5°. Figure III.10 shows the aircraft model response to a commanded 10 ft change in altitude. The aircraft settled in its new

altitude within 15 seconds. Elevator responded in negative direction as expected with about 25% increase in throttle required to execute the climb. The bode plots for both inner and outer loops are shown in Figure III.11.

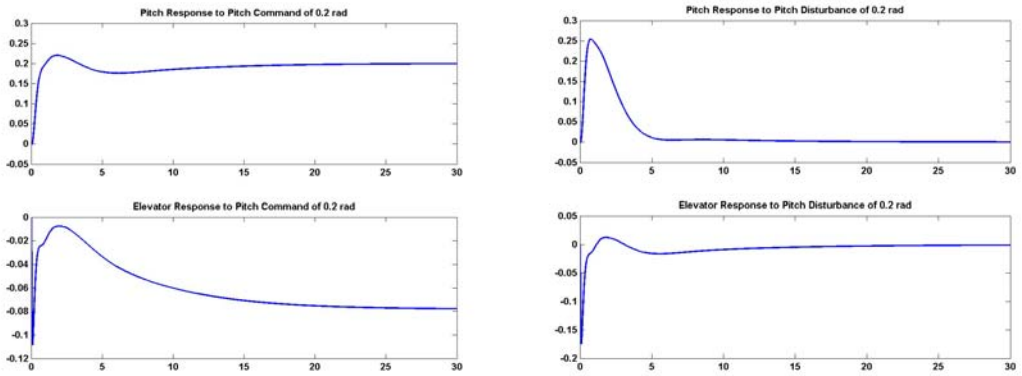


Figure III.9 Pitch Control Loop Responses

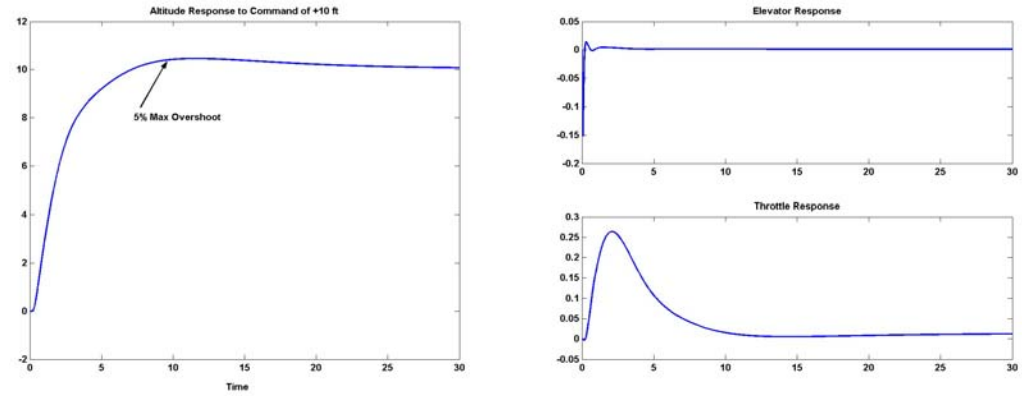


Figure III.10 Altitude Control Loop Responses

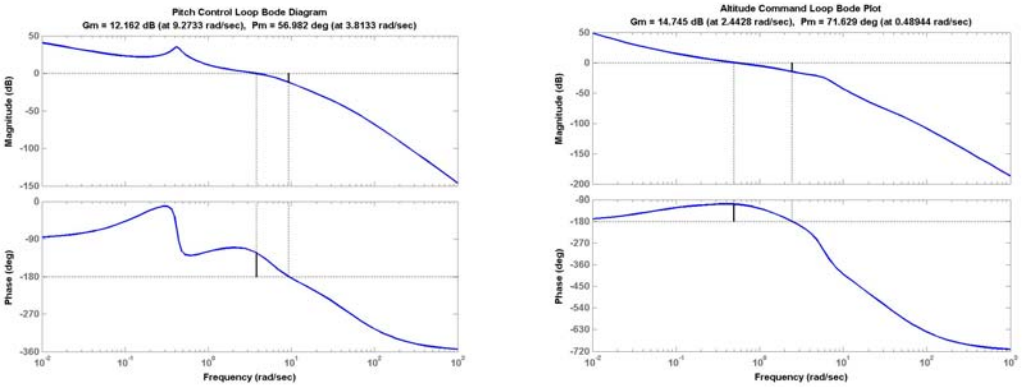


Figure III.11 Altitude Control Inner and Outer Loops Bode Plots

4. Heading Controller

The heading controller was designed in a similar fashion to the altitude controller. The outer loop took commanded heading angle error and produced an angle of bank command. The inner loop took the bank command to generate the required aileron deflection. The inner loop consisted of a PI controller on the bank angle error and a roll rate feedback, while the outer loop consisted of a PI controller to null the heading error. The controller structure is shown in Figure III.12. Figure III.13 shows that the controller was able to track roll commands within 5 seconds and heading commands within 20 seconds.

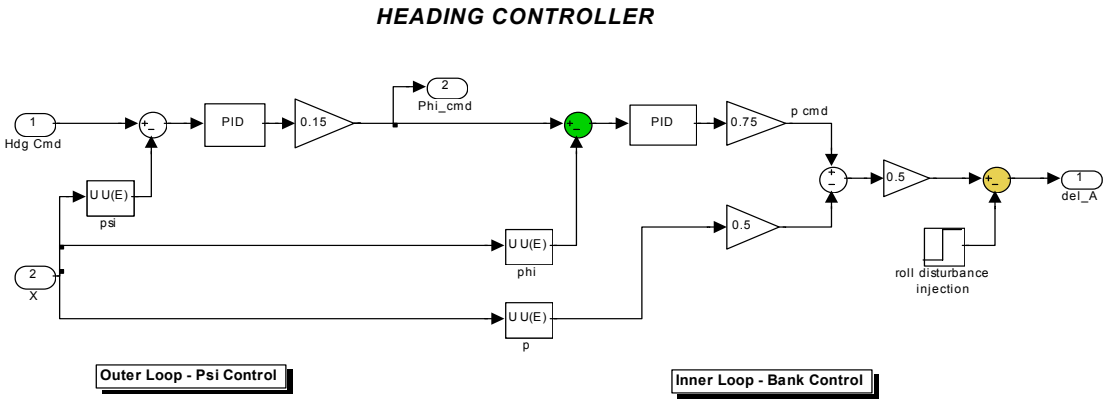


Figure III.12 Heading Controller Block Diagram

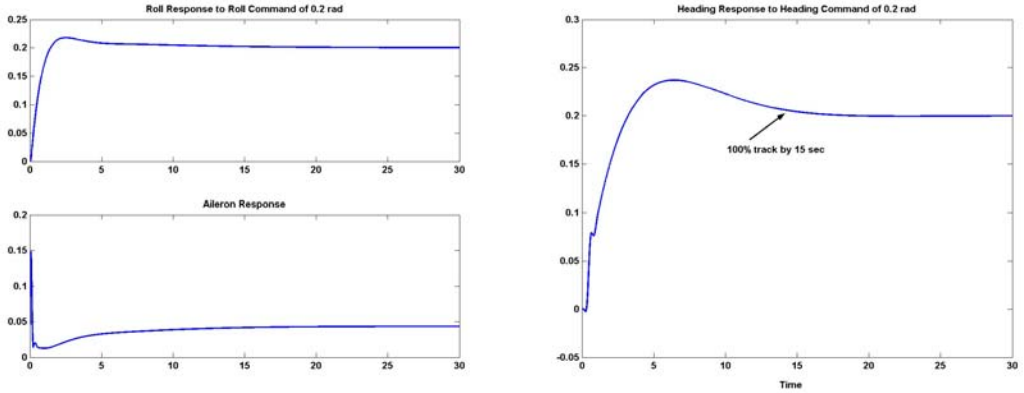


Figure III.13 Roll and Heading Control Loop Responses

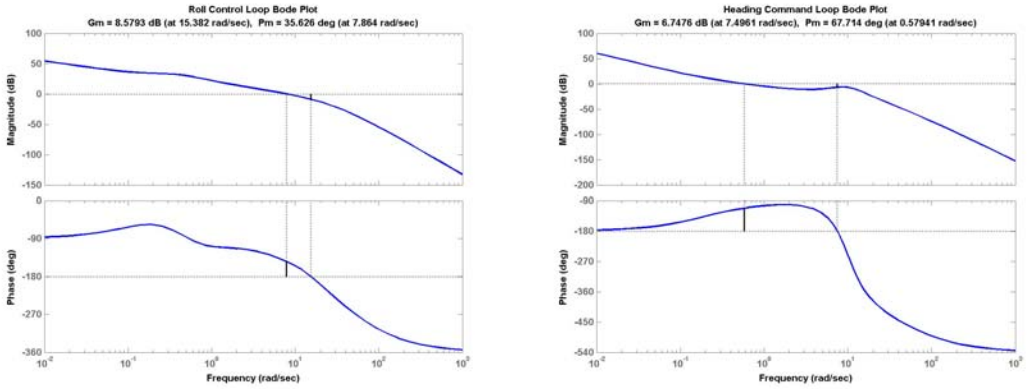


Figure III.14 Heading Control Inner and Outer Loops Bode Plots

5. Complete Controller

The complete controller was implemented as shown in Figure III.15. The gain and phase margins and closed-loop bandwidths are given in Table III.1 below.

Loop	GM (dB)	PM (deg)	CLBW (rad/s)
Yaw Damper	14.2	46.3	30
Vt	Inf	91.8	0.65
Pitch Control	12.2	57.0	9.27
Altitude Commad	14.7	71.6	0.88
Roll Control	8.6	35.6	14.5
Heading Command	6.75	67.7	0.78

Table III.1 Classical Controller Bandwidth Gain and Phase Margins

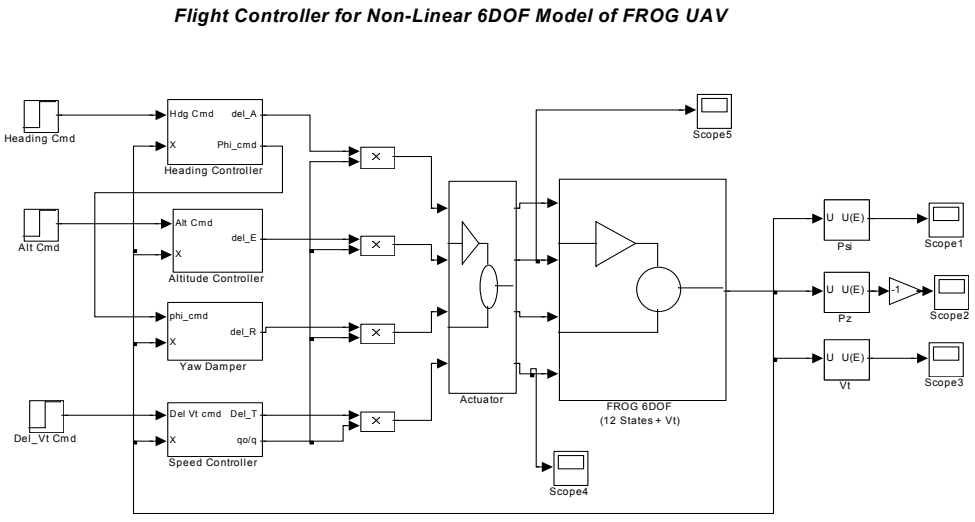


Figure III.15 Complete Flight Controller using Classical Control Design

C. LQR CONTROLLER DESIGN

The second flight controller was designed to control 4 variables in steady-state flight – namely, airspeed (V_t), sideslip (v), heading (ψ) and altitude (h). The design was done using an Integral LQR structure with state-feedback and is based on the technique presented in [8]. The design requirements adopted were:

- Zero steady state errors to constant command in airspeed, sideslip, heading and altitude.
- Overshoot to step commands in altitude and airspeed must not exceed 10%.
- Rise time in response to step altitude commands and step airspeed commands should be around 10 sec.
- Gain margin in control loops should be at least 6 db and phase margins at least 45 degrees.
- Aileron, elevator and rudder loop bandwidth should be around 10 rad/sec. Thrust loop bandwidth should not exceed 5 rad/sec.

The asymptotic property of the LQR controller inherently ensures a stable feedback system. The optimal feedback gain $K_{opt} = R^{-1}B^TP$ generated by the LQR method minimizes the cost function,

$$J = \int_0^\infty (x^{\mathrm{T}}Qx + u^T Ru)\ dt \tag{III.41}$$

and assures that the closed loop system is stable, assuming that (A,B) is stabilizable, and (Q,A) is detectable. The matrices Q≥0, R>0 are weighting matrices, which determine the relative cost of error and energy in the states and control inputs. The P≥0 matrix is the unique stabilizing solution to the Algebraic Riccatti Equation.

$$A^TP + PA - PBR^{-1}B^TP + Q = 0 \tag{III.42}$$

To ensure zero steady state errors to constant altitude and airspeed commands, the integral control is used in conjunction with the LQR technique. The altitude error ($h_c - h$) and the airspeed error ($u_c - u$) were fed to integrators. The integrating action will ensure

that the inputs to each integrator, and hence the altitude and airspeed errors, will be driven to zero in steady state in response to constant commands.

1. Stabilizable and Detectable Criteria

To design the controller using LQR (linear quadratic regulator) technique, the plant must be stabilizable and detectable. A check on the 12-state [$u\ v\ w\ p\ q\ r\ \phi\ \theta\ \psi\ P_x\ P_y\ P_z$] aircraft model for the 4 controlled variables showed that only 10 states where observable. As such, the 10 state equation of motion model with only [$u\ v\ w\ p\ q\ r\ \phi\ \theta\ \psi\ P_z$] was used for controller design using the step as follows.

$$x = [u\ \ v\ \ w\ \ p\ \ q\ \ r\ \ \phi\ \ \theta\ \ \psi\ P_z]^T$$

- Construct the synthesis model for the plant.
- Insert transmission zeros to the synthesis model. (This will be the 'target' poles location for the state-feedback plant.)
- Linearize the synthesis model.
- Adjust the Q and R matrices to vary the cost of states and control inputs. (start with identity).
- Compute the optimal gain K using MATLAB's "lqr(A,B,Q,R,N)".
- Insert the optimal gain K for the plant's states and error states feedback.
- Repeat the last 2 steps while adjusting Q and R to achieve desired control bandwidths, gain and phase margins.

2. Synthesis Model and Controller Structure

The synthesis model and controller structure are shown in the next few figures. The Matlab code created to compute the feedback gains, closed-loop response, bandwidth, gain and phase margins can be found in Appendix C.

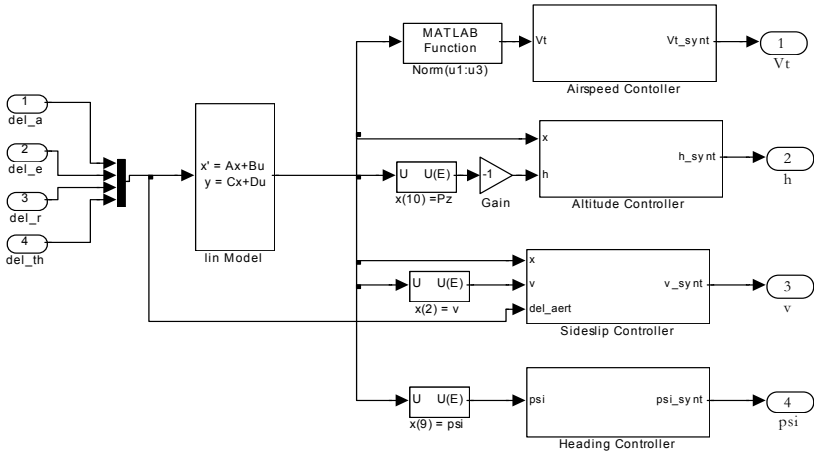


Figure III.16 Overview of Synthesis Model for Controller

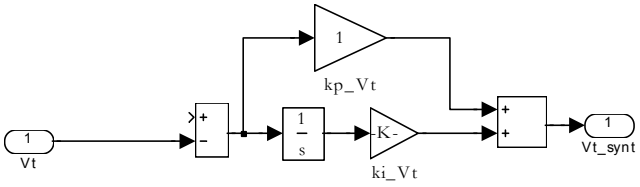


Figure III.17 Creating Real Synthesis Pole

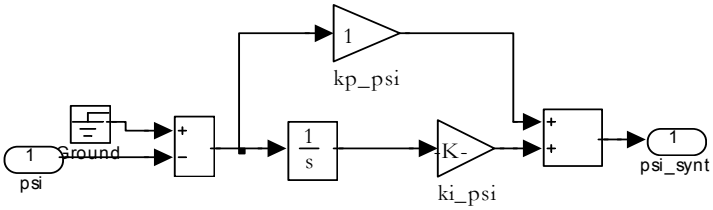


Figure III.18 Creating Complex Synthesis Pole

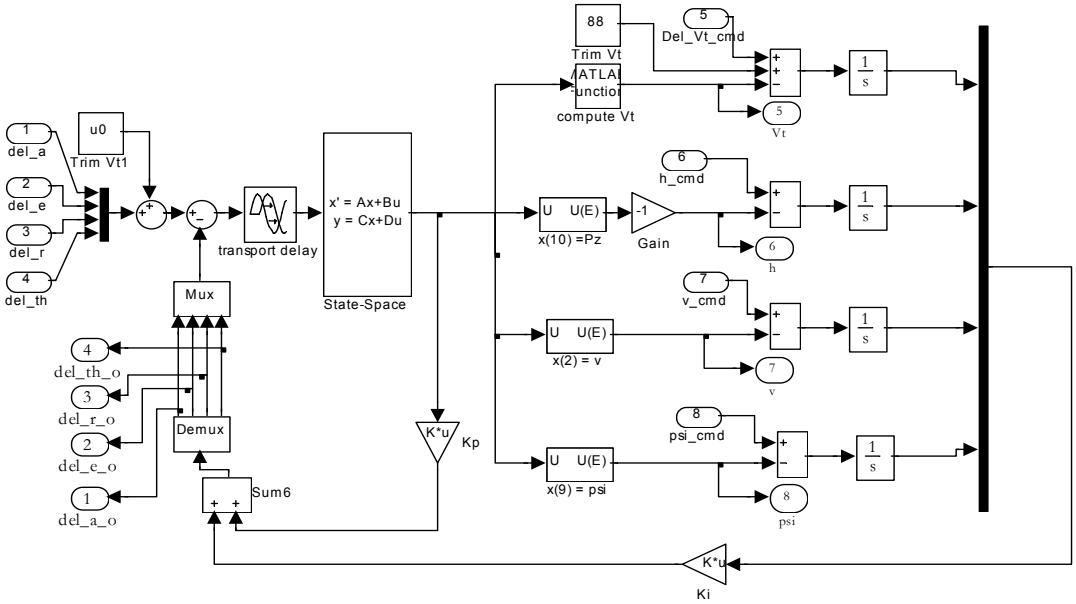


Figure III.19 Linear Integral LQR Controller Structure

For the selected design, two pair of transmission zeros were added in the h-synthesis loop (at $\zeta = 0.9$, wn = 0.4) and v-synthesis loop (at $\zeta = 0.9$, wn = 1.2), and a real pole each for Vt and ψ at 0.7 rad/s and 1.5 rad/s respectively. The diagonal elements of the Q and R matrices were set to identity and [2000,5000,1500,2] respectively. The 3dB closed-loop bandwidths and open-loop gain and phase margins achieved are shown in Table III.2. In comparison with the classical controller, the Integral LQR controller offers better phase margins. However, it had assumed full-state feedback and may not be directly implementable if any of the 10-states listed previously is not easily measurable or tends to fluctuate.

Control Loop	3dB BW (rad/s)	Gain Margin (dB)	Phase Margin (°)
Aileron	4.8	12	78.9
Elevator	8.2	18	52.3
Rudder	1.2	15	91.3
Throttle	4.6	21	69.7

Table III.2 LQR Controller Bandwidth Gain and Phase Margins

3. Complete LQR Controller

The non-linear system simulation structure adopted in Figure III.20 was based on a method described in [9]. The method is based on the observation that the linear controller obtained is designed to operate on the perturbations of the plant's input and outputs about the trimmed condition. Differentiating the measured outputs before they are fed to the gains extracts out the perturbations for which the gains are designed to operate on. To preserve the input-output behavior of the feedback system, a integrator is inserted after the feedback gains are computed. The performance of the LQR controller on the non-linear FROG UAV model thus obtained is shown in Figure III.21.

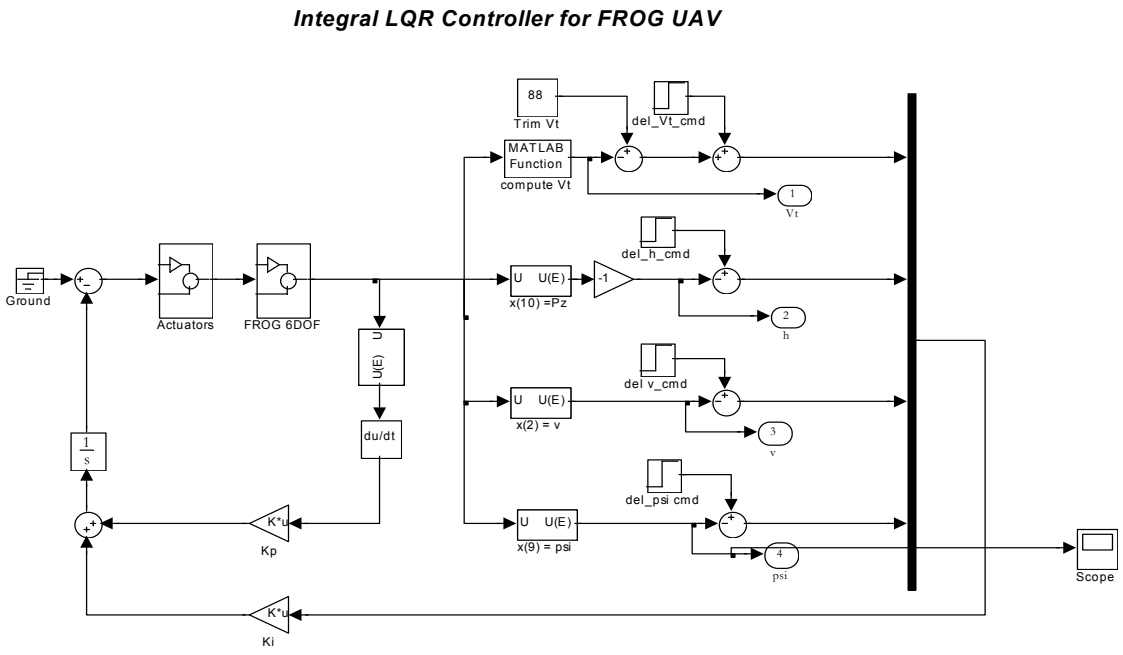


Figure III.20 Non-Linear LQR Controller Implementation

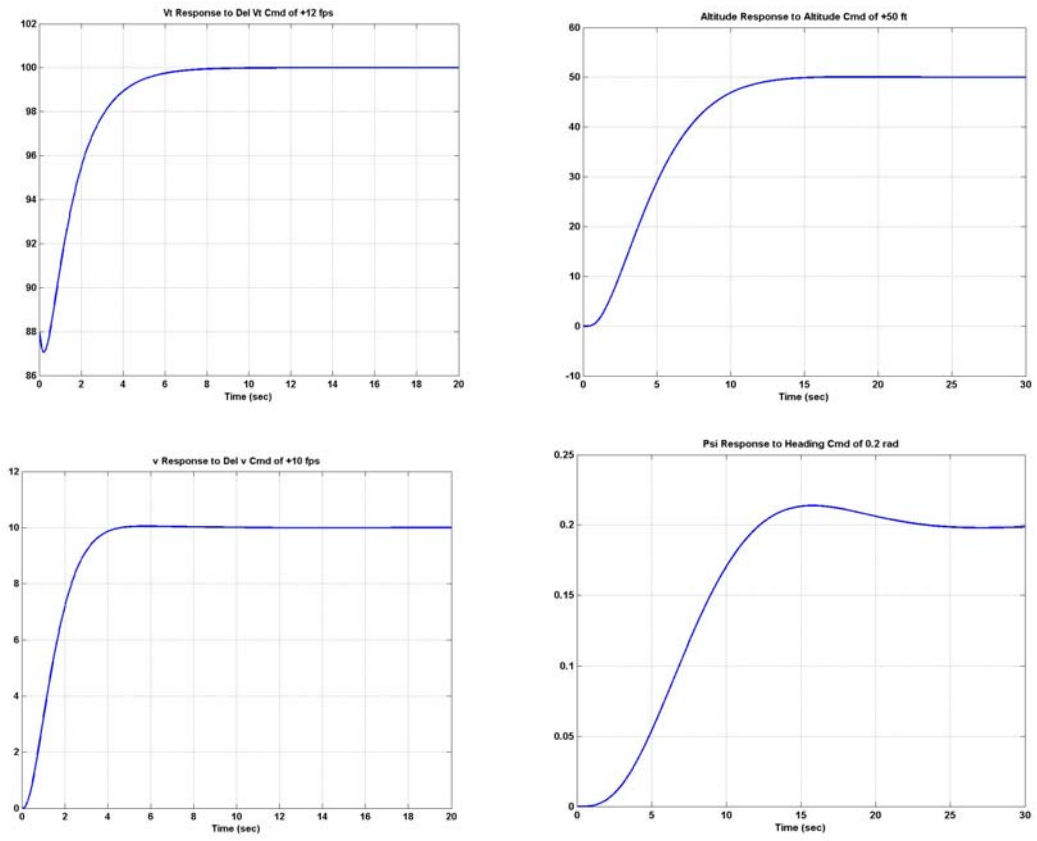


Figure III.21 LQR Controller Performance with Non-linear UAV Model

a) Airspeed Change of 12ft/s b) Altitude Change of 50 ft

c) Sideslip Velocity of 10 ft/s d) Heading Change of 0.2 rad

D. CONTROLLER COMPARISON

Coupling between the longitudinal and lateral control modes was observed on both classical inner-loop-outer-loop controller and integral LQR controller. The controller response and performance can be analyzed based on the sign convention used in [10] and shown in Figure III.22.

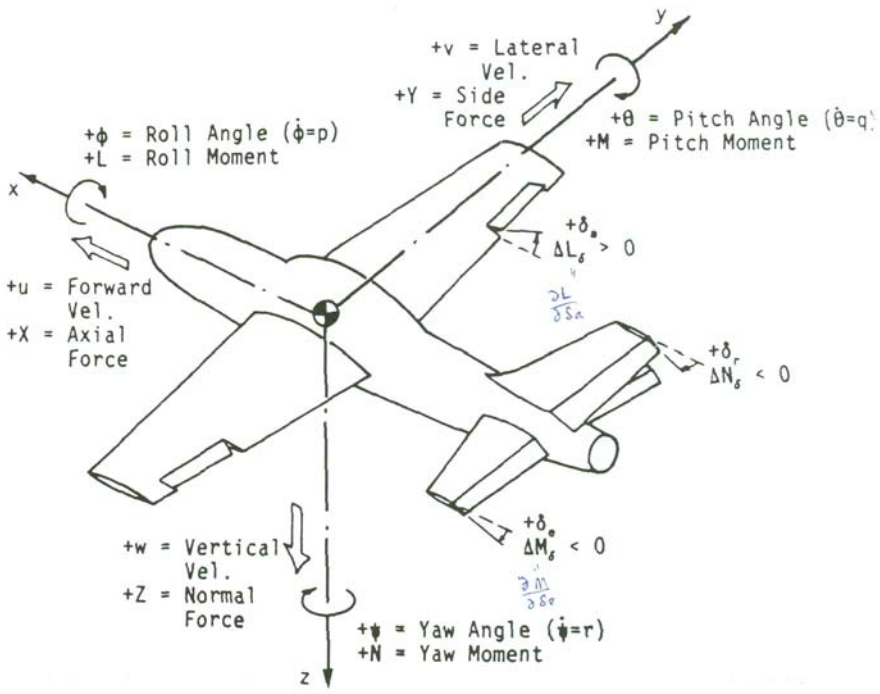


Figure III.22 ANSI/AIAA Sign Convention for Control Surface Deflection [From 10]

In the aircraft with a classical controller, the aircraft's response to an altitude increase of 20 feet in Figure III.23 shows typical longitudinal mode coupling. The controller input a elevator up (negative δ_e) command to tilt the aircraft nose up immediately upon receiving the altitude increase command, airspeed dropped to 85 fps and the throttle had to be increased by about 50% above trim level to bring the airspeed back to 88 fps. Figure III.24 shows the reverse coupling relation. A step command of +12 fps sent to the speed controller causes the throttle to open and aircraft to accelerate. As the aircraft speeds up, it generates more lift and as a result begins to climb by up to 6 feet. The altitude controller immediately commands a positive δ_e to arrest the climb and attempt to return the aircraft to its original altitude. As it returns to the original altitude, the throttle input stabilizes at 20% about trim value to maintain the new airspeed.

Lateral coupling can be observed in Figure III.25. When a step command of +0.2 radian was input to the heading controller, the controller immediately issued a positive aileron control input to bank the aircraft right. The yaw controller simultaneously issued negative δ_r input for the rudder to provide turn coordination as designed.

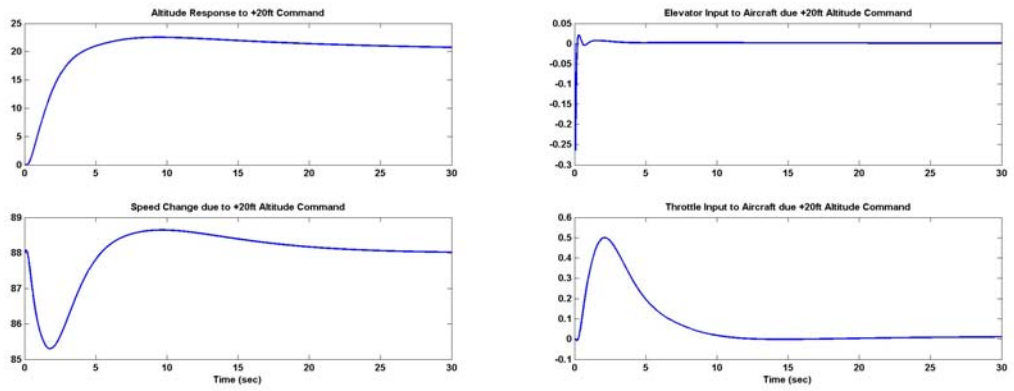


Figure III.23 Classical Controller – Response to Altitude Change of +20 feet

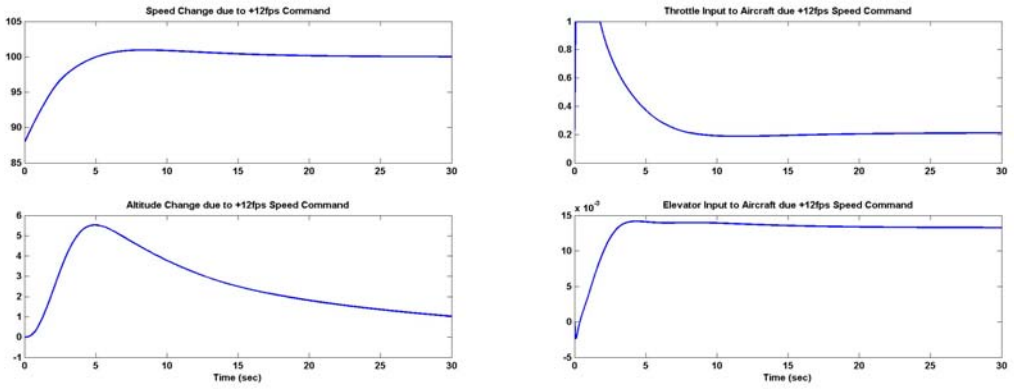


Figure III.24 Classical Controller - Response to Speed Change of +12 fps

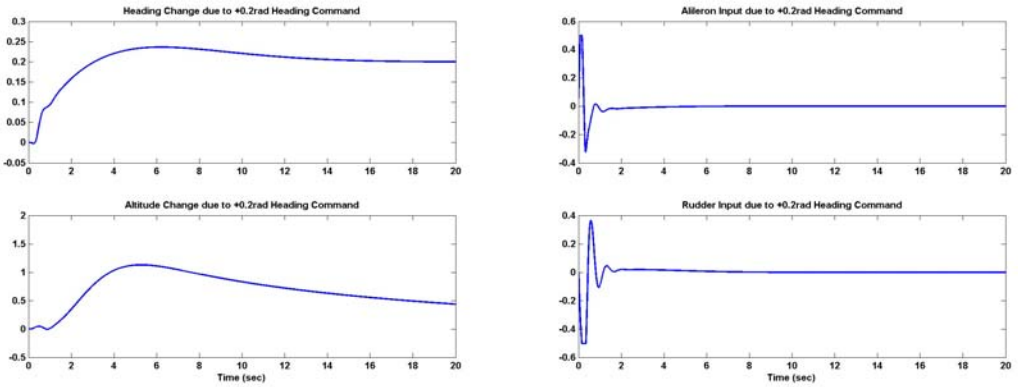


Figure III.25 Classical Controller - Response to Heading Change of +0.2 rad

Similar kind of coupling effects can be observed in the Integral LQR controller. However, with full-state feedback, the integral LQR controller is able to minimize the coupling effects better. For example, as shown in Figure III.26, although the negative δ_e and positive $\delta_{throttle}$ were still issued by the controller to execute the altitude change, the throttle input was issued earlier compared to the classical controller and resulted in only minimal drop in the aircraft speed as the aircraft climbed to its commanded new altitude. The coupling effect on altitude was also much reduced when a speed change was commanded. This is shown in Figure III.27. The altitude changed less than 1 feet in the integral LQR controller aircraft versus 6 feet in the classical controller aircraft.

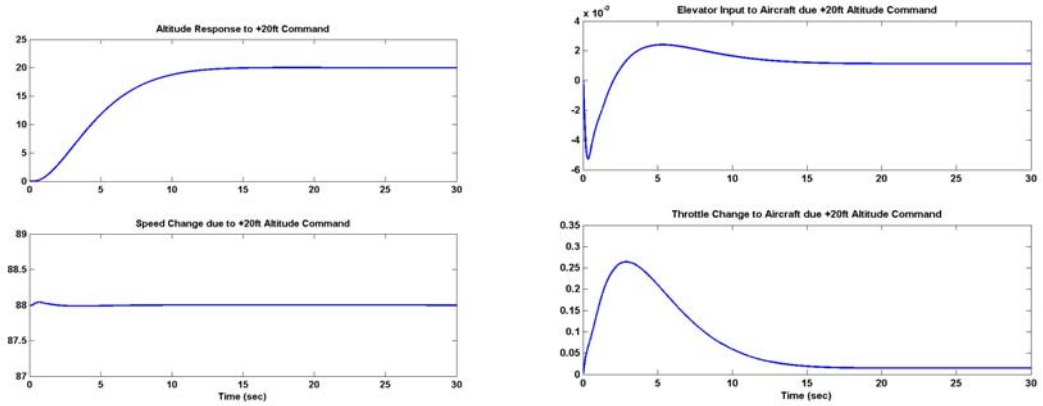


Figure III.26 LQR Controller – Response to Altitude Change of +20 feet

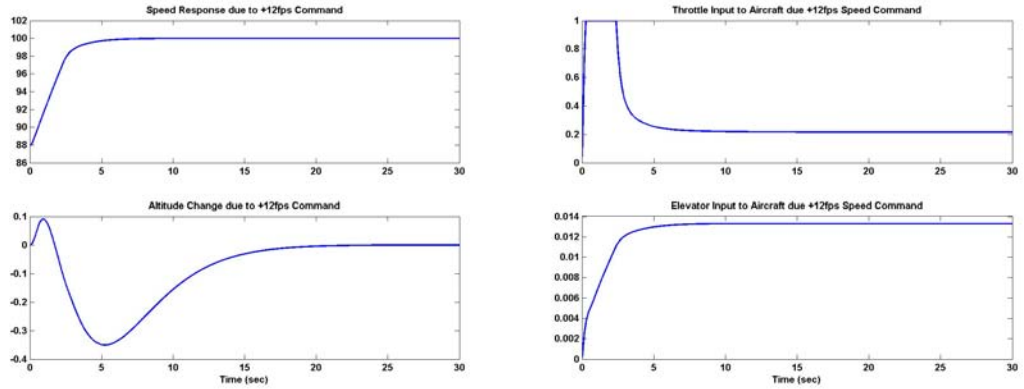


Figure III.27 LQR Controller - Response to Speed Change of +12 fps

Similarly, the lateral coupling effect is much reduced in the integral LQR controller. The coupled altitude increase due to aileron deflection needed to execute the heading change was about 0.2 feet in Figure III.28 compared to that of around 1 foot in the classical controller aircraft in Figure III.25.

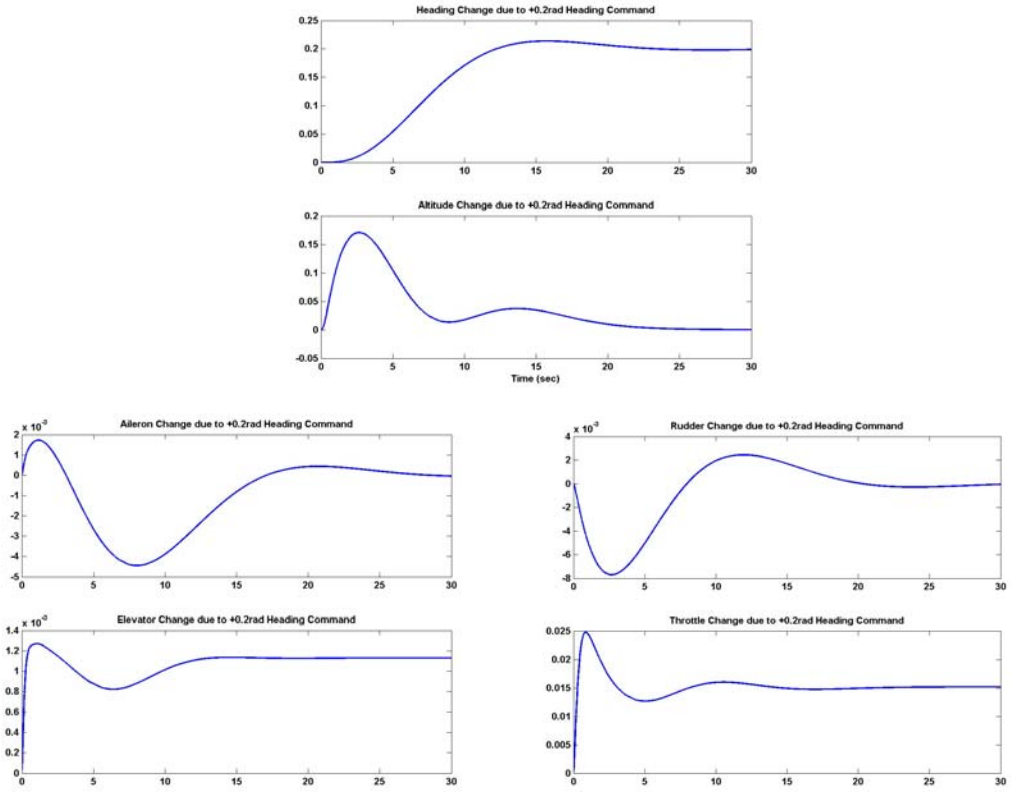


Figure III.28 LQR Controller - Response to Heading Change of +0.2 rad

In comparison, the LQR controller uses all four control inputs to execute commanded changes instead of one control input for each response characteristics as in the classical inner-outer loop design. Hence, it is more responsive and more efficient in control inputs utilization. For example, comparing Figure III.28 and Figure III.25, the aileron and rudder inputs to execute heading change is more than one order smaller in the case of the integral LQR. However, the integral LQR structure is based on full-state feedback and assumes all the states are measurable and not too noisy. Also, the cross-coupled nature of the control inputs makes it more difficult to deduce the response of each control input in a given flight situation and makes troubleshooting more difficult.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. NAVIGATION ALGORITHM DESIGN

This section discusses the development of an infrared vision-based shipboard navigation algorithm to determine the position and orientation of an aircraft with respect to a ship with three visible points of known separation. It covers the problem formulation and includes a simulation example based on the numerical analysis technique proposed in [11] to determine the range of an aircraft which is instrumented with an IR camera with respect to the ship. The simulation serves as a precursor to explore implementation of an autonomous shipboard landing algorithm for the FROG UAV.

A. SHIPBOARD LANDING PROBLEM FORMULATION

Design of the autonomous shipboard landing algorithm requires first determining the range and orientation of the aircraft to a ship which has a minimum of three identifiable points. However, using only three reference points (RPs) always results in more than one solution as has been shown by a number of researchers of this problem. This non-uniqueness is usually resolved at close ranges by using more than three points but for the purposes of this study, it was assumed that three reliable points may be computed from the location of the smokestack and the extents (width and height) of the ship even when the aircraft is sufficiently far from the ship as an input condition to the navigation algorithm development. Figure IV.1 shows an example of such a scenario that can be used by the algorithm developed.

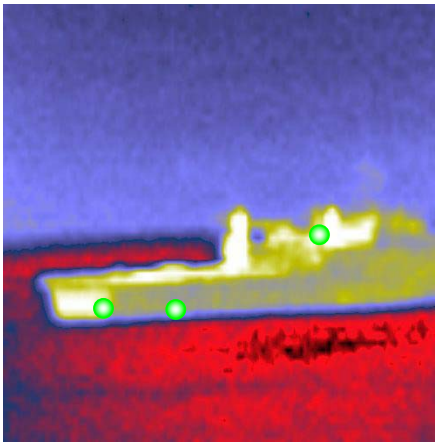


Figure IV.1 Examples showing images of three RPs

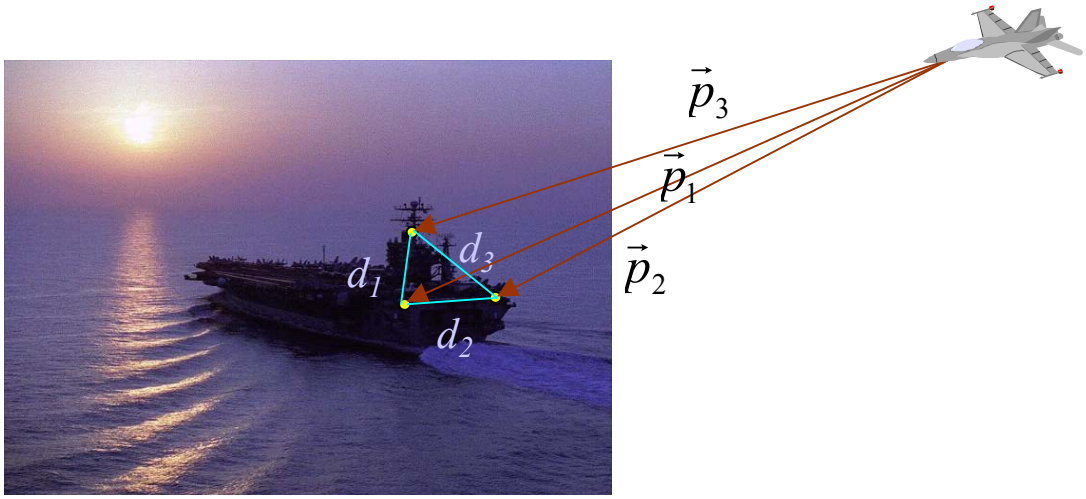


Figure IV.2 The 3-point geometry applied to shipboard navigation

Once the image of the three RPs is established, the geometry of the Posed 3-Point (P3P) problem to determine the range and orientation of the aircraft's IR camera with respect to the ship in the shipboard autolanding navigation sceneario is shown in Figure IV.3 formulates the problem formulation as follows: Let $\vec{p}_i = \{x_i, y_i, z_i\}$, $i = 1,...,3$ denote the vectors connecting the origin of the camera frame O with the three known points P_i, $i = 1,...,3$. Let d_i, $i = 1,...,3$ denote distances between these points. Then,

$$\|\vec{p}_1 - \vec{p}_2\| = d_1 \neq 0,\ \|\vec{p}_1 - \vec{p}_3\| = d_2 \neq 0,\ \|\vec{p}_2 - \vec{p}_3\| = d_3 \neq 0,\ d_1 \neq d_2 \neq d_3, \tag{IV.1}$$

and $s_i = \|\vec{p}_i\|$, $i = 1,...,3$ denote the norms of the vectors $\vec{p}_i$. Using the pinhole camera model, the projection of each RP onto the image plane of the camera with the focal length f has the following form:

$$\pi(p_i) = \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \frac{f}{x_i} \begin{pmatrix} y_i \\ z_i \end{pmatrix},\ i = 1,...,3. \tag{IV.2}$$

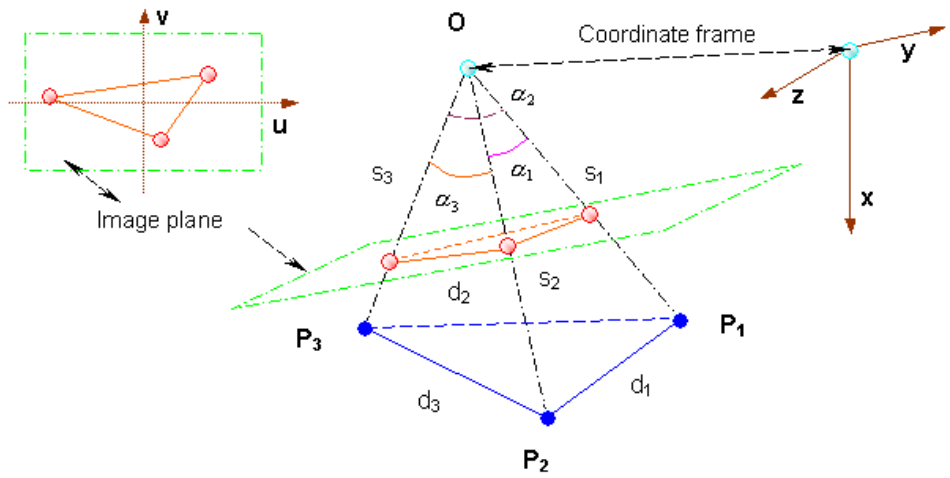


Figure IV.3 Three-point perspective pose estimation problem geometry

Combining equations (IV.1) and (IV.2) result in nine equations with nine unknowns $\{x_i, y_i, z_i\}$, $i = 1,...,3$. Using (IV.2),

$$y_i = \frac{x_i u_i}{f}, \; z_i = \frac{x_i v_i}{f}. \tag{IV.3}$$

Substituting these expressions into the (IV.1), equations (IV.2) and (IV.1) can be reduced to a set of three nonlinear equations in three unknowns:

$$\begin{aligned}
&\sum_{i=1,2}\left(f^2 + u_i^2 + v_i^2\right)x_i^2 - 2\left(f^2 + u_1u_2 + v_1v_2\right)x_1x_2 = \left(fd_1\right)^2,\\
&\sum_{i=1,3}\left(f^2 + u_i^2 + v_i^2\right)x_i^2 - 2\left(f^2 + u_1u_3 + v_1v_3\right)x_1x_3 = \left(fd_2\right)^2,\\
&\sum_{i=2,3}\left(f^2 + u_i^2 + v_i^2\right)x_i^2 - 2\left(f^2 + u_2u_3 + v_2v_3\right)x_2x_3 = \left(fd_3\right)^2,
\end{aligned} \tag{IV.4}$$

To simplify the notation, (IV.4) can be re-written as follows

$$\begin{aligned}
&Ax_1^2 - 2D_{12}x_1x_2 + Bx_2^2 = \bar{d}_1,\\
&Ax_1^2 - 2D_{13}x_1x_3 + Cx_3^2 = \bar{d}_2,\\
&Bx_2^2 - 2D_{23}x_2x_3 + Cx_3^2 = \bar{d}_3.
\end{aligned} \tag{IV.5}$$

where the coefficients A, B, C, $\bar{d}_i$, $i = 1,...,3$ are strictly positive by construction.

Using (IV.5) one can obtain another system of equations better suited for further analysis. First, observe that

$$x_1 = \frac{f}{\sqrt{A}} s_1,\ x_2 = \frac{f}{\sqrt{B}} s_2,\ x_3 = \frac{f}{\sqrt{C}} s_3. \tag{IV.6}$$

Rewriting system (IV.5) in terms of s_i, $i = 1,...,3$ gives

$$\begin{aligned} s_1^2 - 2s_1 s_2 \cos\alpha_1 + s_2^2 &= d_1^2, \\ s_1^2 - 2s_1 s_3 \cos\alpha_2 + s_3^2 &= d_2^2, \\ s_2^2 - 2s_2 s_3 \cos\alpha_3 + s_3^2 &= d_3^2, \end{aligned} \tag{IV.7}$$

where $\cos\alpha_1 = \dfrac{(\vec{p}_1, \vec{p}_2)}{\|\bar{p}_1\|\|\bar{p}_2\|}$, $\cos\alpha_2 = \dfrac{(\vec{p}_1, \vec{p}_3)}{\|\bar{p}_1\|\|\bar{p}_3\|}$, $\cos\alpha_3 = \dfrac{(\vec{p}_2, \vec{p}_3)}{\|\bar{p}_2\|\|\bar{p}_3\|}$ (from Figure IV.3)

The system of equations in (IV.7) has an upper bound of eight (2^x2^x2) real solutions. Moreover they form four symmetric pairs, because if a triplet (s_1^*, s_2^*, s_3^*) is a solution, than the triplet $(-s_1^*, -s_2^*, -s_3^*)$ forms a solution as well. Geometrically, (IV.7) can be described as an intersection of three orthogonal elliptic cylinders with the semiaxes rotated around corresponding symmetry axes by the angle of 45º. This follows directly from the canonical form of equation (IV.7). The magnitudes of the semiaxes for each cylinder are equal to

$$a_i, b_i = \frac{d_i}{\sqrt{1 \pm \cos\alpha_i}},\ i = 1,...,3. \tag{IV.8}$$

It is clear that the intersection of any two cylinders is always non-empty and the number of solutions in this case is infinite. However, by adding a third cylinder one can get only a finite number of intersection points. In practice for the system (IV.7), this number cannot be zero or two. Therefore, the only possible set of solutions contains four, six or eight points.

If Assumption (A1) is made such that camera is always in front of the plane defined by three RPs $P_i,\ i=1,...,3$, the x-component of each vector $\vec{p}_i=\{x_i,y_i,z_i\}$, $i=1,...,3$ that satisfies this assumption must be positive (i.e. $s_i>0$, $i=1,...,3$).

With the substitution in (IV.9) and (IV.10), the system of equations in (IV.7) can be reduced to a fourth-order polynomial in $s_i^{\,2}$.

$$u^*=\frac{s_2}{s_1}\ \text{and}\ v^*=\frac{s_3}{s_1} \tag{IV.9}$$

$$s_2=u^*+s_1\cos\alpha_1\ \text{and}\ s_3=v^*+s_1\cos\alpha_2, \tag{IV.10}$$

The numerical analysis technique in [11] further showed that with assumptions

A$_2$) $\underset{i=1,3}{\underline{min}}\, s_i >> \underset{i=1,3}{\underline{max}}\, d_i$;

A$_3$) the camera is sufficiently far from the ship.

$$0<\alpha_i<\pi/2,\quad \sum_{i=1}^{3}\alpha_i<\pi,\quad \alpha_i\le\sum_{\substack{l=\overline{1,3}\\ l\neq i}}\alpha_l,\ i=1,...,3. \tag{IV.11}$$

And from first two equations in (IV.7), s_2 and s_3 can be expressed as:

$$s_i=\cos\alpha_{i-1}s_1\pm\sqrt{\left(\cos\alpha_{i-1}s_1\right)^2-\left(s_1^2-d_{i-1}^2\right)},\ i=2,3. \tag{IV.12}$$

With the consequence that all possible solutions for s_1 that satisfies assumptions A1-A3 is bounded by the interval:

$$0<s_1\le s_1^*=\min_{i=1,2}\left\{\frac{d_i}{\sqrt{1-\cos^2\alpha_i}}\right\}=\min_{i=1,2}\left\{\frac{d_i}{\sin\alpha_i}\right\}. \tag{IV.13}$$

B. AIRCRAFT-SHIP ORIENTATION DETERMINATION

Based on the analysis presented in [11], the following algorithm was applied to solve the P3P. Suppose a good initial guess of $\vec{n}^{(0)}$ the normal to the plane generated by the three points is available, then for step *k*:

i) Solve numerically equation (IV.10) for $x_1^{(k)}$ in the interval equation (IV.13), using $x_1^{(k-1)}$ as an initial guess;

$$s_2 = u^* + s_1 \cos\alpha_1 \text{ and } s_3 = v^* + s_1 \cos\alpha_2, \tag{IV.10}$$

$$0 < s_1 \le s_1^* = \min_{i=1,2}\left\{\frac{d_i}{\sqrt{1-\cos^2\alpha_i}}\right\} = \min_{i=1,2}\left\{\frac{d_i}{\sin\alpha_i}\right\}. \tag{IV.13}$$

ii) substitute each solution $x_1^{(k)}$ obtained in *i)* into (IV.3) to get $\hat{\vec{p}}_{i\_1}^{(k)}$ and $\hat{\vec{p}}_{i\_2}^{(k)}$;

$$y_i = \frac{x_i u_i}{f},\ z_i = \frac{x_i v_i}{f}. \tag{IV.3}$$

iii) compute normals

$$\vec{n}_1^{(k)} = \frac{(\hat{\vec{p}}_{1\_1}^{(k)} - \hat{\vec{p}}_{2\_1}^{(k)}) \times (\hat{\vec{p}}_{1\_1}^{(k)} - \hat{\vec{p}}_{3\_1}^{(k)})}{\left\|\hat{\vec{p}}_{1\_1}^{(k)} - \hat{\vec{p}}_{2\_1}^{(k)}\right\| \left\|\hat{\vec{p}}_{1\_1}^{(k)} - \hat{\vec{p}}_{3\_1}^{(k)}\right\|} \text{ and } \vec{n}_2^{(k)} = \frac{(\hat{\vec{p}}_{1\_2}^{(k)} - \hat{\vec{p}}_{2\_2}^{(k)}) \times (\hat{\vec{p}}_{1\_2}^{(k)} - \hat{\vec{p}}_{3\_2}^{(k)})}{\left\|\hat{\vec{p}}_{1\_2}^{(k)} - \hat{\vec{p}}_{2\_2}^{(k)}\right\| \left\|\hat{\vec{p}}_{1\_2}^{(k)} - \hat{\vec{p}}_{3\_2}^{(k)}\right\|};$$

iv) choose set $\hat{\vec{p}}_{i\_1}^{(k)},\ i = \overline{1,3}$ or $\hat{\vec{p}}_{i\_2}^{(k)},\ i = \overline{1,3}$ that maximizes the dot product $\left\langle \vec{n}^{(k)}, \vec{n}^{(k-1)} \right\rangle$.

Using the solution provided by the P3P algorithm the relative orientation of the aircraft with respect to the plane formed by the three RP's can be computed as follows:

Let *{3p}* denotes an orthogonal coordinate system attached to the plane generated by the three RP's, let *{c}* denotes the coordinate system attached to the camera and let

${}^{c}_{3p}R$ be the coordinate transformation from *{3p}* to *{c}*. Form three orthogonal vectors $\vec{r}_1$, $\vec{r}_2$, $\vec{r}_3$ using the correct solution $\hat{\vec{p}}_1$, $\hat{\vec{p}}_2$, $\hat{\vec{p}}_3$ as follows:

$$\vec{r}_1 = \frac{(\hat{\vec{p}}_2 - \hat{\vec{p}}_1)}{\left\|\hat{\vec{p}}_2 - \hat{\vec{p}}_1\right\|},\ \vec{r}_3 = \frac{(\hat{\vec{p}}_2 - \hat{\vec{p}}_1)\times(\hat{\vec{p}}_3 - \hat{\vec{p}}_1)}{\left\|\hat{\vec{p}}_2 - \hat{\vec{p}}_1\right\|\left\|\hat{\vec{p}}_3 - \hat{\vec{p}}_1\right\|},\ \vec{r}_2 = \vec{r}_3 \times \vec{r}_1. \tag{IV.14}$$

Then ${}^{c}_{3p}R = \begin{bmatrix}\vec{r}_1 & \vec{r}_2 & \vec{r}_3\end{bmatrix}$.

But from geometry, the transformation matrix ${}^{c}_{3p}R$ can also be expressed using Euler angles as:

$${}^{c}_{3p}R = \begin{bmatrix} \cos\psi_{3p}\cos\theta_{3p} & \sin\psi_{3p}\cos\theta_{3p} & -\sin\theta_{3p} \\ \cos\psi_{3p}\sin\theta_{3p}\sin\phi_{3p} - \sin\psi_{3p}\cos\phi_{3p} & \sin\psi_{3p}\sin\theta_{3p}\sin\phi_{3p} + \cos\psi_{3p}\cos\phi_{3p} & \cos\theta_{3p}\sin\phi_{3p} \\ \cos\psi_{3p}\sin\theta_{3p}\cos\phi_{3p} + \sin\psi_{3p}\sin\theta_{3p} & \sin\psi_{3p}\sin\theta_{3p}\cos\phi_{3p} - \cos\psi_{3p}\sin\phi_{3p} & \cos\theta_{3p}\cos\phi_{3p} \end{bmatrix}, \tag{IV.15}$$

where ψ_{3p}, θ_{3p}, ϕ_{3p} are yaw, pitch and bank angles, respectively, with respect to the plane formed by the three RP's. Therefore, the Euler angles can be found in the following manner:

$$\psi_{3p} = \arctan\frac{r_{12}}{r_{11}},\ \theta_{3p} = -\arcsin r_{13},\ \phi_{3p} = \arctan\frac{r_{23}}{r_{33}}. \tag{IV.16}$$

In general, the coordinate system *{3p}* does not coincide with the inertial coordinate system *{i}*. Therefore, the attitude *{c}* of the camera with respect to *{i}* can be found using (IV.16)from the transformation matrix ${}^{c}_{3p}R\,{}^{3p}_{i}R$, where ${}^{3p}_{i}R$ can be obtained from the known positions of the three RP's in *{i}*, using the same manner.

C. ALGORITHM SIMULATION

The P3P algorithm developed above was applied to determine the range of the aircraft with respect to the ship in a simulation example described below.. The ship is moving North at a constant speed of 10*m/s*. Its motion is characterized by pitch and heave oscillations with a period of 12*sec*. The aircraft is performing a left turn with descent from the initial point (-1450, -200, 470)*m* with respect to the ship's initial position at an

airspeed of $53m/s$. The camera's focal length is $f = 0.1m$ and declination angle with respect to a/c longitudinal axis is -6*deg*. The errors in the projection of each RP onto the image plane of the camera are modeled as independent Gaussian random process with zero mean and standard deviation of one pixel.

Figure IV.4 shows the horizontal projection of each of the three RP's on the ship tracked by the camera and of the aircraft's motion. Figure IV.5 gives the corresponding 3D representation.

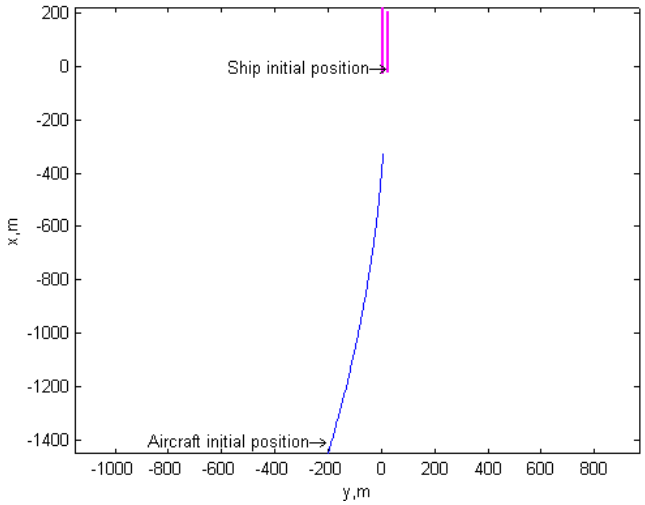


Figure IV.4 Horizontal projection of a/c's and ship's motion

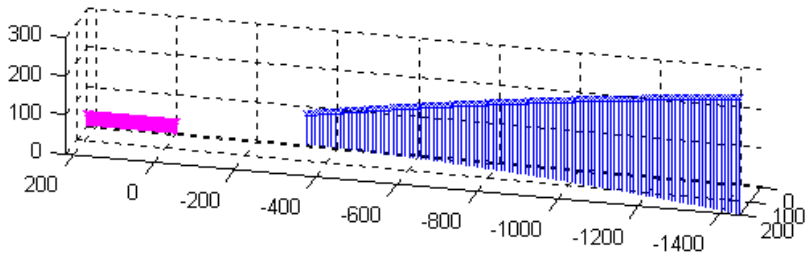


Figure IV.5 3D representation of the simulation scenario

In summary, simulation results shows that the proposed algorithm is a feasible method of implementing flight navigation and trajectory tracking in order to adopt certain flight profiles at various stages of the mission. This can be further demonstrated on the FROG UAV test-bed in the near future.

V. GROUND TEST RESULTS

To test capability of the computer in capturing GPS, IMU and PWM data on aircraft in the same manner it would operate in flight, a full system verification test was conducted at the UAV Lab. All the aircraft equipment and the newly installed miniature computer were powered from an internal battery. A floppy was used to boot up the xPC operating kernel and then removed for the UAV nose fairing to be installed. The data collection application was 'built' on the Host PC and uploaded to the aircraft's computer via wireless serial modem. Commands to start and stop data collection were issued from the Host PC remotely. The engine was started on several tests to check for electromagnetic interference (EMI) on data transfer and to assure pilot control capability from various distances.

A. DATA ANALYSIS

The ground test proved that the aircraft computer was able to correctly collect and interpret the data from various instruments. The GPS, IMU, PWM and A/D data captured on a particular data collection test is presented in the next few pages. Figure V.1 to V.9 show the plots of GPS RMC and GGA data captured when the UAV was moved in a triangular path of approximately South-West, followed by East, followed by North in front of the UAV Lab.

1. GPS Signals

From the GPS RMC data captured, Figure V.1 shows the correct UTC, while Figures V.2 and V.3 show the latitude and longitude of Monterey airport area where the test was conducted. The data in Figures V.2 and V.3 was combined and re-presented in ground coordinates to show the path taken by the aircraft while data capturing was in progress. This is shown in Figure V.4 together with the bearing (215°, followed by 80°, followed by 350°) the UAV had taken along the path. Figure V.5 shows the ground speed at which the UAV was moved around the path and the date of the test (i.e. 13/03/02).

Figure V.6 shows the magnetic variation in degrees and the direction of magnetic north and true north.

> [t(35:55),RMC_	RMC_UTC(35:55)]
0. 0975	181853.00
0. 1000	181853.00
0. 1025	181853.00
0. 1050	181853.00
0. 1075	181853.00
0. 1100	181853.00
0. 1125	181853.00
0. 1150	181853.00
0. 1175	181854.00
0. 1200	181854.00
0. 1225	181854.00
0. 1250	181854.00
0. 1275	181854.00

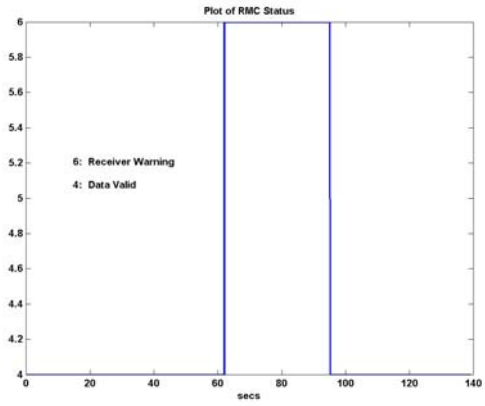


Figure V.1 GPS RMC UTC (left) and RMC Status (right)

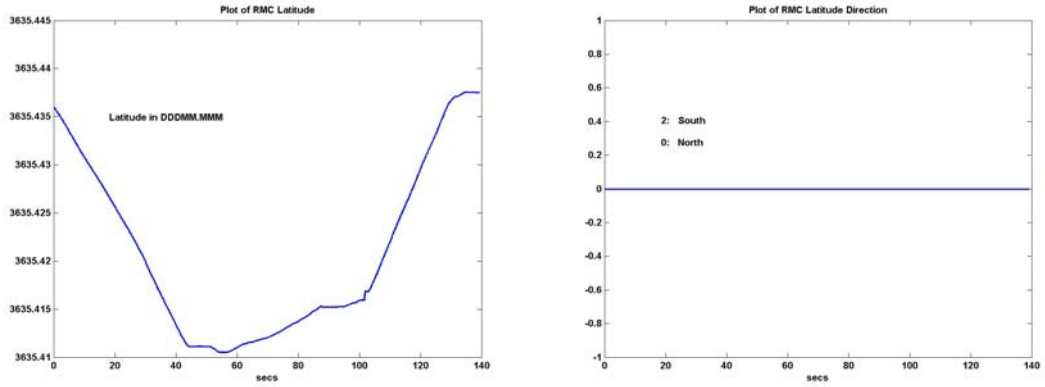


Figure V.2 GPS RMC Lattitude (left) and Latitude Direction (right)

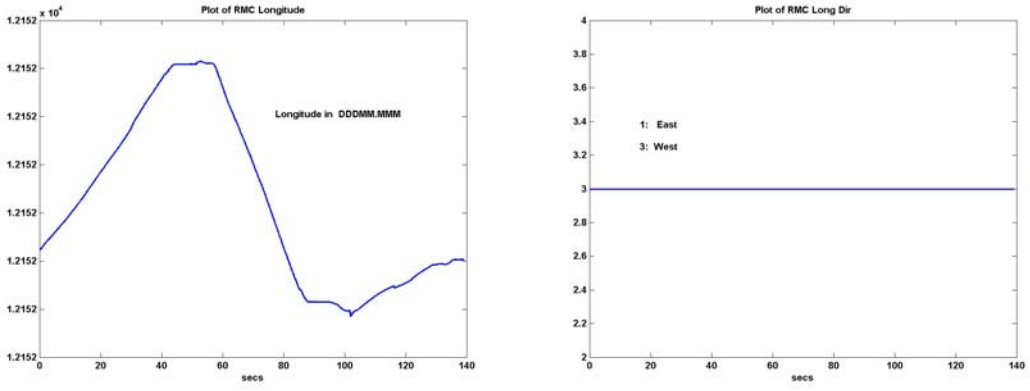


Figure V.3 GPS RMC Longitude (left) and Longitude Direction (right)

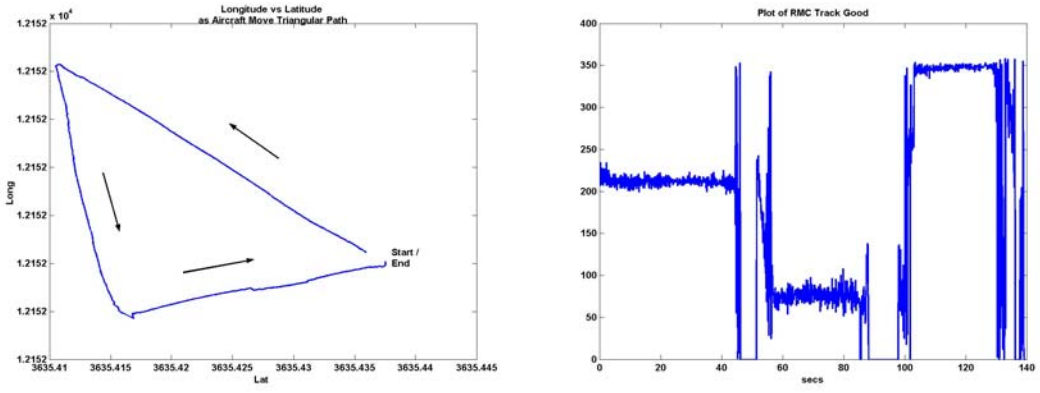


Figure V.4 Position Plot From RMC data (left) and GPS RMC Track (right)

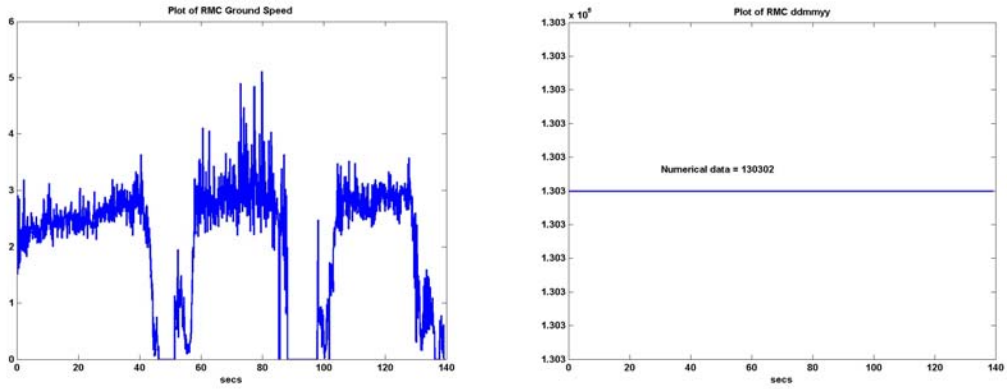


Figure V.5 GPS RMC Ground Speed (left) and dd/mm/yy (right)

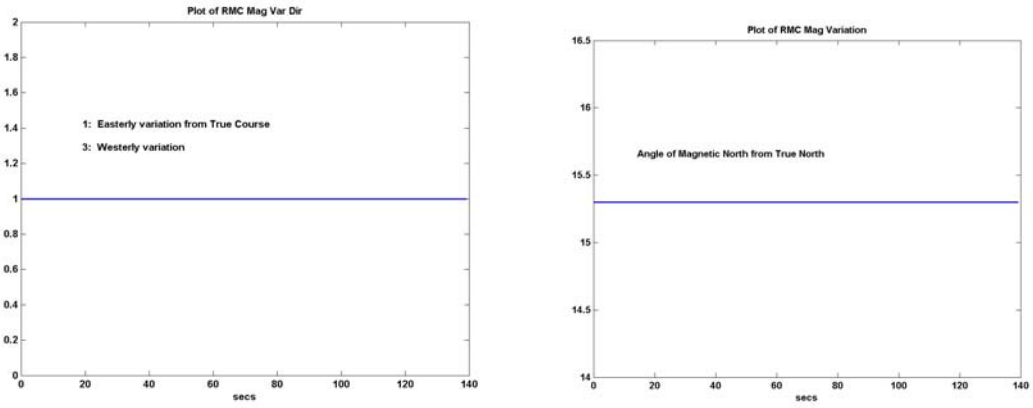


Figure V.6 GPS RMC Magnetic Variation (left) and MV Angle (right)

The GPS GGA sentence provides data complementary to the GPS RMC sentence for navigation purposes. Figures V.7 shows the UTC time in the hh.mm.ss.ss format and the number of satellites in use. Figure V.8 demonstrates a clear correspondence to the data in Figure V.7 (right) and shows that the horizontal dilution of precision increased to around 2.0 when the number of available satellites in sight decreased from 5 to 4, and to 3.6 when the number of available satellites dropped further to 3. Figure V.9 shows the altitude of antenna in meters above mean sea-level.

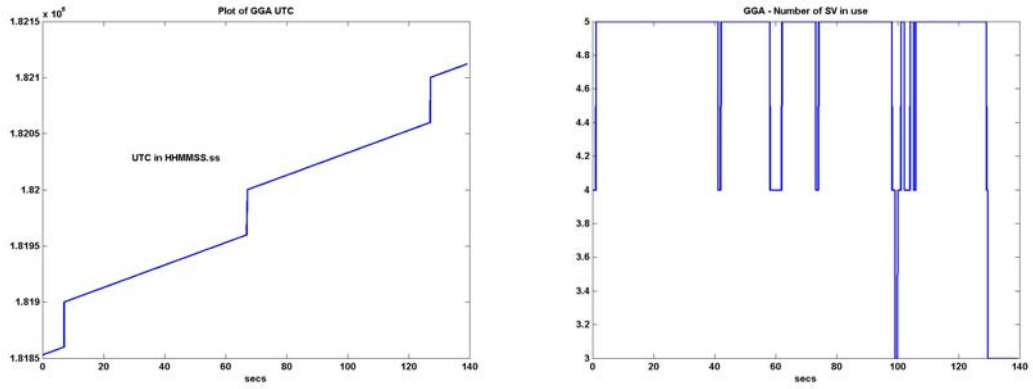


Figure V.7 GPS GGA UTC (left) and Number of Satellite Vehicles Used (right)

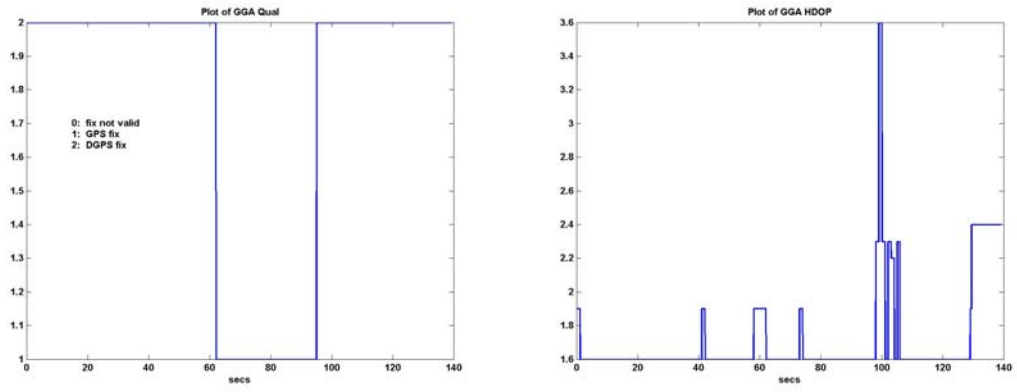


Figure V.8 GPS GGA Fix Quality (left) and HDOP (right)

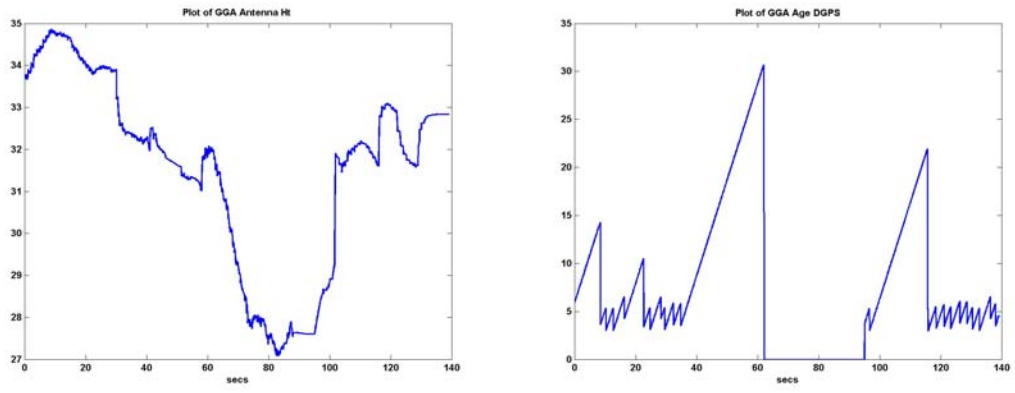


Figure V.9 GPS GGA Antenna Height (left) and DGPS Data Age (right)

2. A/D and PWM Signals

The voltages of the control surfaces and throttle servos captured by the A/D signal are presented together with the corresponding PWM commands received by the Futaba® receiver in Figures V.10 to V.13. Each pair of figures clearly shows servos responding to the commands issued to it. While the A/D measured signals are almost noise-free, the PWM signals captured showed data drop-outs occasionally over a period of 5 to 10 seconds. As such, some form of filtering needs to be implemented if the PWM captured data is required. In our case, this does not pose a serious concern as the PWM signal captured is not used for flight control processing. Rather, it is measured to record the commands sent to the Futaba® receivers and correlate with how the aircraft responds when the Futaba® controller is used.

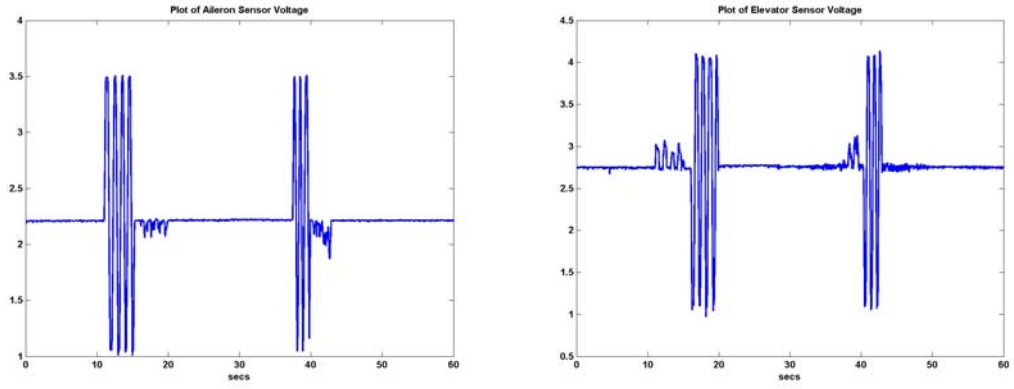


Figure V.10 Aileron and Elevator Servo Voltages measured by A/D

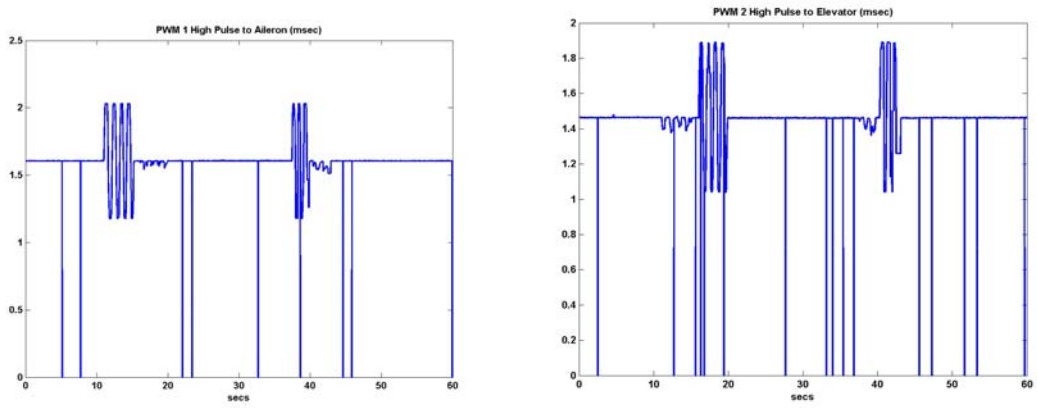


Figure V.11 PWM Commands Issued to Aileron and Elevator Servos

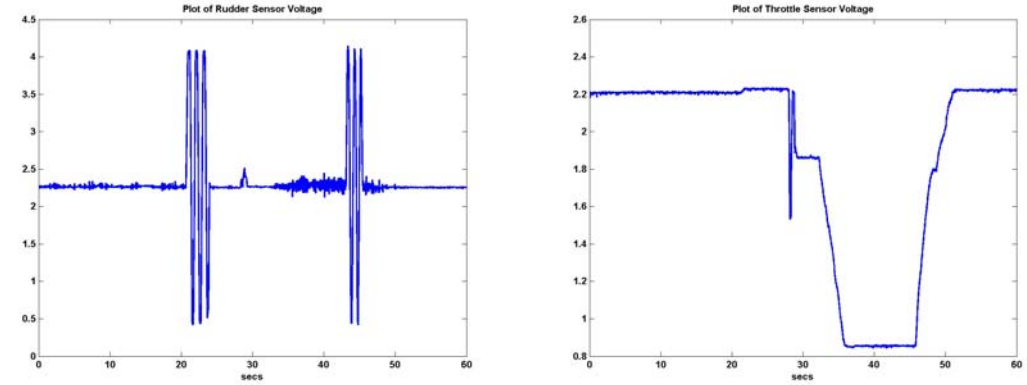


Figure V.12 Rudder and Throttle Servo Voltages measured by A/D

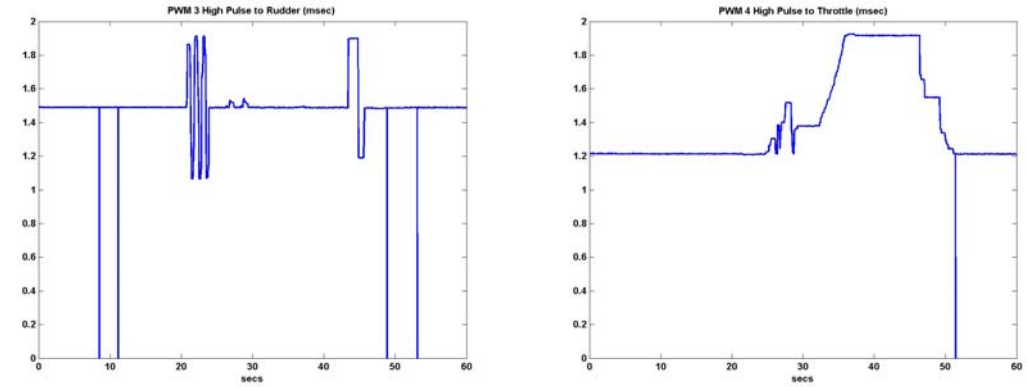
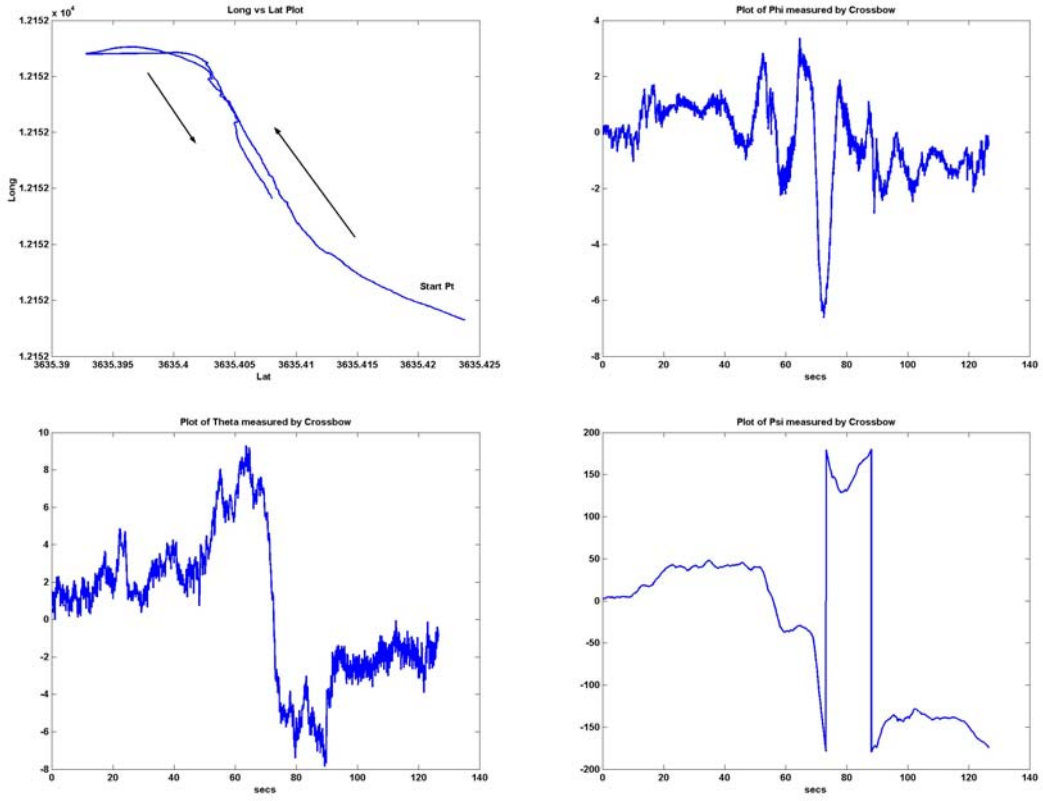


Figure V.13 PWM Commands Issued to Rudder and Throttle

3. Crossbow Signals

The signals from the Crossbow IMU were captured on another ground test after Crossbow IMU was initialized. They are presented as time-plots in Figure V.14. The path taken as the UAV was pushed on its landing gear around the NPS UAV Lab is given in the first plot. The ± 2° roll angle measured is in-line with the manner in which the body of the aircraft tilts as it moves along its path, except when it reached the slightly sloped turnaround point when the left-wing was slightly lower as the aircraft turns. The effect of the slight ground slope near the turnaround point was evident in the pitch angle plot. The pitch angle increases from 0 to 7° as the aircraft slowly moves up the slope and changes to –7° as it turns back. The longitude versus latitude plot shows the aircraft initially veered right with respect to its initial heading as it moves away from the UAV Lab. It subsequently turned left for some distance before making an about turn. This is clearly reflected in the Psi (heading) data measured by Crossbow.



The angular rates and linear accelerations recorded by the Crossbow are more 'noisy' but still correlate well with the FROG UAV's motion. Roll rate and pitch rate fluctuate around zero as the UAV moves around since the landing gear introduces vibration to the Crossbow. In the yaw-rate data captured, the about-turn at about 70 seconds correspondent to the left turn the aircraft had to take. The acceleration in the z-axis shows +1g due to gravity as expected.

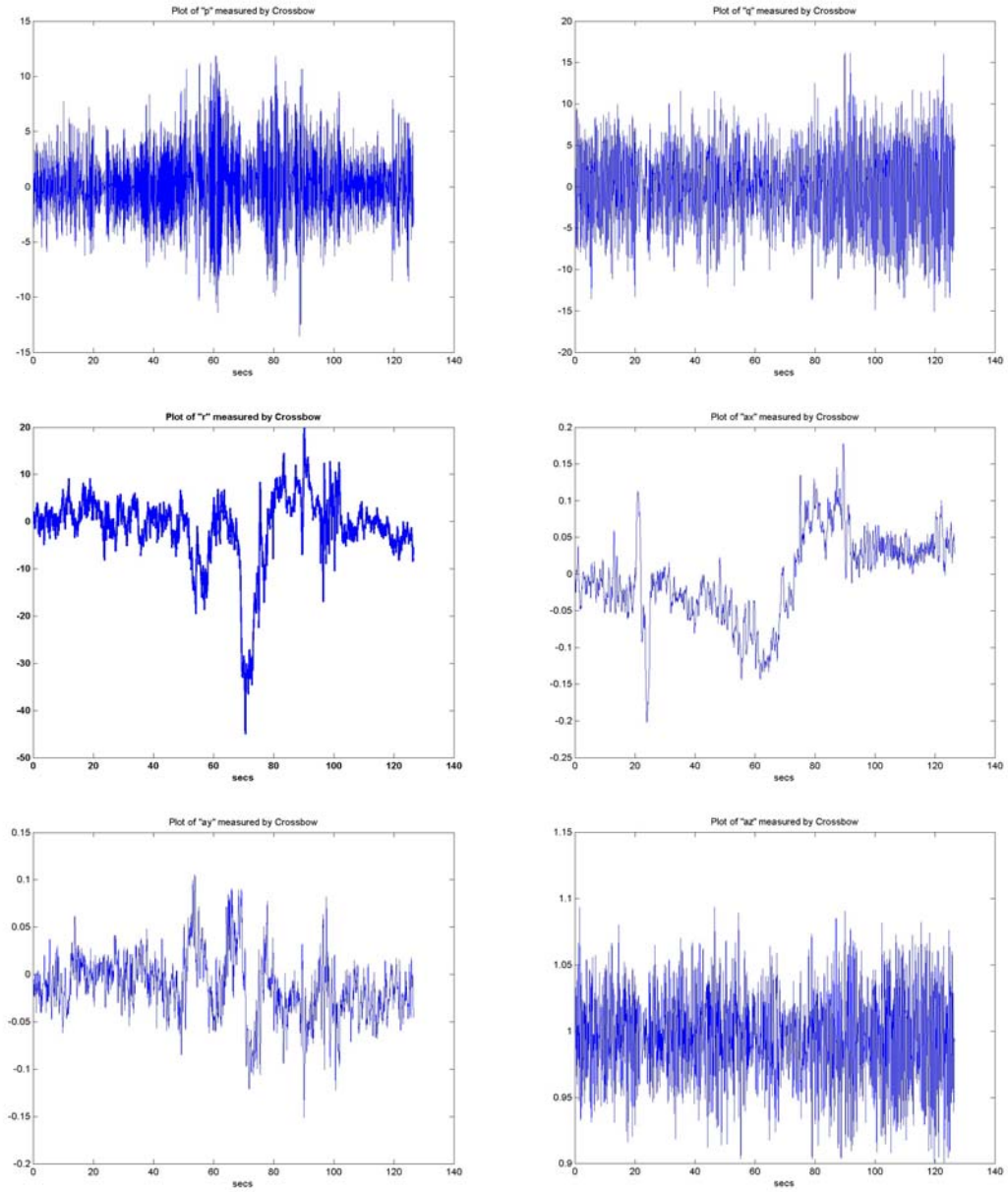


Figure V.14 Crossbow IMU Signals

B. EMI ISSUES

Despite successful demonstration of sensor data capture and interpretation on the new onboard computer, the ground tests revealed two communication related problems in the setup. The first was that data downlink from Target PC to Host PC via the Freewave® modems was intermittent. The xPC real-time operating system would display a "COM failure" message on the Host PC during some downloads. The second was that during data downloading, the aircraft control surfaces respond to commands sent from the Futaba® remote control with significant delay. This is unacceptable as it can result in a total loss of control when emergency pilot mode of control is used. Both problems occur only when the wireless modem is used in place of a direct null-modem cable between the Host PC and Target PC.

Initial investigations explored the effect of changing some of the modem's transmission parameters such as maximum data packet size, minimum data packet size, power setting, the transmission frequency and baud rate but failed to resolve the occasional COM failure problem that occurred during data download. Subsequent investigations narrowed down to electromagnetic interference (EMI) generated on the aircraft and serial communication data latency timings as possible causes.

The first assessment resulted from the observation that upload of application programs and commands to the aircraft computer did not encounter transmission problems when radio signals were sent to the onboard modem. However, during the downloading of recorded flight data, when the onboard modem was transmitting, EMI from cables onboard the aircraft can corrupt the transmitted data and result in transmission errors. This hypothesis was supported by absence of communication failure when the PWM capture cable was removed.

Another possible cause could be the delay time the xPC operating kernel expects data to be available on the serial link during transmission. This may introduce communication failure in the xPC software depending on how much latency time the xPC operating kernel would accommodate. In a direct cable link, such a problem does not arise as the cable is always connected and does not require transmission handshaking as in the wireless modems.

The freezing of control surface response on the aircraft when the onboard modem is transmitting could be due to the interference of the modem transmission on the Futaba® receiver.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

In support of the research objective on coordination and control of a cluster of UAVs, a miniature computer critical for onboard real-time data processing in order to execute autonomous guidance and control of aircraft was constructed and installed on the FROG UAV in this project. In contrast to the original setup where flight and navigation data had to be transmitted to a computer on the ground for processing and control commands computation before being re-transmitted back to the aircraft for control execution, the new onboard computer integrates all the sensor data for control and guidance commands computation on the aircraft. This reduces sensor data processing time from 25 milliseconds (at 40 Hz downlink data rate) by 2 orders of magnitude to around 350 microseconds. It would also reduce control commands execution time to tens of microseconds given the computer generates PWM signals to control the servos directly as compared to 160 milliseconds in the current setup. The presence of an onboard computer also provides the computational capability to implement data processing for more advanced research activities in the future. To date, the computer and all software drivers written to interface with various onboard instruments have been tested.

Additionally, a new 6-DOF model of the FROG UAV has been developed in Simulink. This will facilitate future simulation of the control and guidance algorithm for the FROG UAV. As part of this thesis, two possible autopilot designs – one using classical control techniques and one using modern control theory – have also been designed for the UAV. A vision-based navigation algorithm for the FROG UAV has also been demonstrated in software simulation. These control and navigation algorithms can be hardware-in-the-loop tested using available facilities at NPS before verification flights in the near future.

The use of the newly marketed xPC rapid prototyping system in this project greatly expedited the development and implementation of the desired control setup. A thorough understanding of the capabilities and ways to overcome its shortcomings has resulted from this project. The expertise accumulated will be useful for many projects at

NPS, not just in the Aeronautics Department, since the rapid prototyping approach can be applied to shorten the design-to-full-system testing timeframe of any conceivable system being developed.

B. RECOMMENDATIONS

With the software for navigation and flight data processing on the aircraft fully verified, the only remaining obstacle preventing a flight test pertains to the wireless communication EMI issue. The following can be pursued in the near future.

1. Measure the signal-to-noise level and the signal power level during the uplink and the downlink to quantify the signal power required and the noise limits the control setup can accommodate.
2. Determine the components causing EMI and introduce EMI filters accordingly to remove or reduce the EMI on the aircraft.
3. Consult Mathworks, Inc on the serial communication latency time the xPC operating kernel can accept and find a way to change this parameter in the xPC operating kernel to accommodate any delay in the serial modem transmission.
4. Explore the use of a wireless Ethernet link in avoiding the interference on the downlink transmission. A wireless Ethernet link can also reduce massive flight data download times by 1 to 2 orders of magnitude.

APPENDIX A. DESCRIPTION OF FROG UAV

BAI TERN

Type
Multipurpose, semi-expendable UAV.

Development
The TERN (Tactical Expendable Remote Navigator) was originally designed and developed by H-Cubed Corporation of Columbia, Maryland. The programme was acquired by BAI Aerosystems in 1993. Capable of a variety of applications, it is recoverable in peacetime training missions but of sufficiently low cost that it could be discarded after a battlefield mission if necessary. On 1 October 1992, a TERN set up an endurance record in FAI Class F3a for unmanned aircraft of 33 hours 39 minutes 15 seconds. A straight-line distance record in the same class was set on 28 September 1993 with a flight of 245.80 n miles (455.23 km; 282.87 miles).

Airframe
Pod-and-boom fuselage with high-mounted wing, sweptback fin and rudder, and low-set tailplane; wing fitted with flaps; GFRP/epoxy construction. Engine mounted above wing centre-section; fixed tricycle landing gear.

Mission payloads
Can include colour TV camera, thermal imager or hazardous agent sensors. One variant **(TERN-C)** has carried a remote IR sensor for aerial detection of chemical warfare agents. Another **(FOG-R)** has been fitted with an Optelecom Fibre Optic DataLink (FODL) and flown non-line of sight 'over and below the hill' for uplink command of the UAV and downlink video imagery. The latter version is stated to be proof against jamming.

Guidance and control
UHF (400 MHz) radio command uplink for remote control; D-band (1.8 GHz) video downlink with data sideband. Options include programmable autopilot, GPS navigation and fibre optic datalink. Wing flaps can be trimmed to enable slow flight speeds for surveillance, or fully extended to assist landing in restricted spaces.

Transportation
Wings detach for containerised storage and transportation.

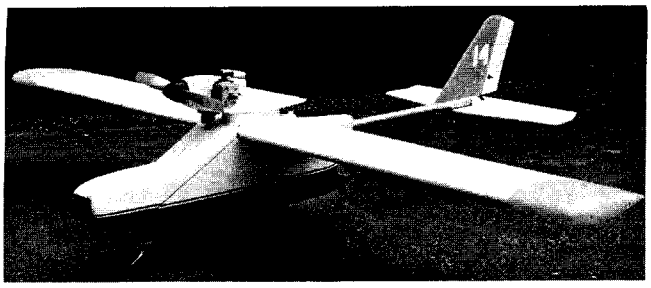
Launch
Conventional wheeled take-off.

Recovery
Conventional wheeled landing.

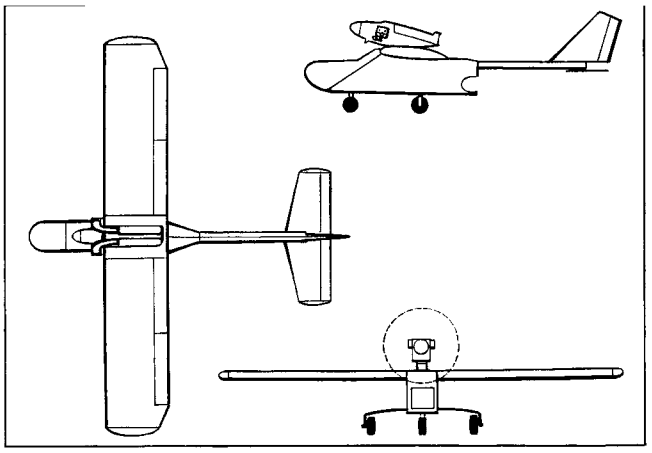
Operational status
In production.

Customers
Has been used by US Army as surrogate fibre optic guided missile (FOG-M) and as an NBC sensor vehicle. Also used by Naval Weapons Center and ARDEC.

Prime contractor
BAI Aerosystems Inc, Easton, Maryland.



BAI TERN semi-expendable multirole UAV 1998/0001584



TERN air vehicle three-view (Jane's/John W Wood) 1998

Power plant
One 7.5 kW (10 hp) 150 cc two-cylinder two-stroke engine (type not known); two-blade fixed-pitch wooden propeller.

Dimensions

Wing span	3.10 m (10 ft 2.0 in)
Wing area	1.57 m² (16.94 sq ft)
Length overall	2.48 m (8 ft 1.5 in)
Height overall	0.86 m (2 ft 10.0 in)
Wheel track	0.74 m (2 ft 5.0 in)

Weights

Weight empty	17.7 kg (39.0 lb)
Max payload	13.6 kg (30.0 lb)
Max T-O weight	34.0 kg (75.0 lb)

Performance

Max level speed	70 kt (129 km/h; 80 mph)
Normal cruising speed	48-52 kt (88-96 km/h; 55-60 mph)
Loiter speed	35-43 kt (64-80 km/h; 40-50 mph)
Stalling speed	31 kt (57 km/h; 35 mph)
* Range	8.6 n miles (16 km; 10 miles)
Typical endurance at above cruising speed	3 h

* *Extendable in autonomous flight mode*

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. AIRBORNE COMPUTER I/O INTERFACE

The hardware setting for the miniature computer and various I/O boards that was built for the UAV is given in the sections below. Similar information from the original AC-104 computer is also given in case the miniature computer needs to be re-configured back to the AC-104 configuration.

1. I/O ADDRESS

The I/O addresses of resources in the new miniature computer are shown in Table B.1. The I/O resource addresses for the original AC-104 is also shown as reference.

AC-104 Computer	Base Address (Offset)	New Computer
IDE hard disk interface	0x1F0	Not Installed
Reserved	0x1F8	Not Installed
Ruby-MM 8 Channel D/A, 24 Chan. Dig. I/O	0x200 (0x40)	Removed
AIM/16-1 16 Channel A/D, 16 Chan. Dig. I/O	0x280 (0x20)	AIM/16-1
Serial Port Number 4 (ET Models Only)	0x2E8	COM4
Reserved	0x2F0	Not Installed
Serial Port Number 2	0x2F8	COM2
Flex/104 IP Carrier Board 1	0x300	Removed
Pause and Error Light Control Register	0x310	Available. Not Used
Flex/104 IP Carrier Board 2	0x320	Not Installed
M-Systems 4MB Flash Disk	0x330	4MB Flash Disk
Ruby-MM-16 16-channel D/A (optional)	0x340	Not Installed
NAI-5718 Synchro/Resolver Board	0x355	Not Installed
Flex/104 IP Carrier Board 3	0x360	Not Installed
Multimode Parallel Port (when enabled)	0x378	Available. Not Used
Ethernet	0x380	Ethernet
Monochrome display adapter	0x3B0	Not Installed
Flat Panel/CRT VGA display adapter	0x3C0	VGA Display
Video Controller	0x3D0	Video Controller
Serial Port Number 3 (ET models only)	0x3E8	COM3
Reserved	0x3F0	Not Installed
Floppy disk controller ports	0x3F2	Floppy disk controller
Serial Port Number 1	0x3F8	0x08
	0x210	QMM-10 Timer/Counter (for PWM capture)
	0x240	QMM-5 Timer Counter (for PWM generation)

Table B.1 I/O Resource Addresses

2. I/O BOARD PIN OUT CONNECTION AND USAGE

The pin utilization for the I/O cards installed in the miniature computer is given in the Tables B.2, B.3 and B.4 below. Bold items indicated connected lines.

AC104	Signal	Board	Board	Signal	AC104
1	In 1	1	2	In 2	26
2	Gate 1	3	4	Gate 2	27
3	**Out 1 (PWM1 out)**	**5**	**6**	**Out 2 (PWM2 out)**	**28**
4	In 3	7	8	In 4	29
5	Gate 3	9	10	Gate 4	30
6	**Out 3 (PWM3 out)**	**11**	**12**	**Out 4 (PWM4 out)**	**31**
7	In 5	13	14	Out 5	32
8	Gate 5	15	16	Frequency Output	33
9	NC	17	18	NC	34
10	NC	19	20	NC	35
11	NC	21	22	NC	36
12	NC	23	24	NC	37
13	NC	25	26	NC	38
14	NC	27	28	NC	39
15	NC	29	30	NC	40
16	NC	31	32	Interrupt Input	41
17	Digital Out 7	33	34	Digital In 7	42
18	Digital Out 6	35	36	Digital In 6	43
19	Digital Out 5	37	38	Digital In 5	44
20	Digital Out 4	39	40	Digital In 4	45
21	Digital Out 3	41	42	Digital In 3	26
22	Digital Out 2	43	44	Digital In 2	27
23	Digital Out 1	45	46	Digital In 1	28
24	Digital Out 0	47	48	Digital In 0	29
25	+5V	49	50	**Ground**	**50**

Table B.2 QMM-5 (PWM Generation) Pin Interface

AC104	Signal	Board	Board	Signal	AC104
1	In 1	1	2	In 2	26
2	**Gate 1 (PWM1 In)**	**3**	**4**	**Gate 2 (PWM1 In)**	**27**
3	Out 1	5	6	Out 2	28
4	In 3	7	8	In 4	29
5	**Gate 3 (PWM2 In)**	**9**	**10**	**Gate 4 (PWM2 In)**	**30**
6	Out 3	11	12	Out 4	31
7	In 5	13	14	Out 5	32
8	Gate 5	15	16	Frequency Output	33
9	In 6	17	18	In 7	34
10	**Gate 6 (PWM3 In)**	**19**	**20**	**Gate 7 (PWM3 In)**	**35**
11	Out 6	21	22	Out 7	36
12	In 8	23	24	In 9	37
13	**Gate 8 (PWM4 In)**	**25**	**26**	**Gate 9 (PWM4 In)**	**38**
14	Out 8	27	28	Out 9	39
15	In 10	29	30	Out 10	40
16	Gate 10	31	32	Interrupt Input	41
17	Digital Out 7	33	34	Digital In 7	42
18	Digital Out 6	35	36	Digital In 6	43
19	Digital Out 5	37	38	Digital In 5	44
20	Digital Out 4	39	40	Digital In 4	45
21	Digital Out 3	41	42	Digital In 3	26
22	Digital Out 2	43	44	Digital In 2	27
23	Digital Out 1	45	46	Digital In 1	28
24	Digital Out 0	47	48	Digital In 0	29
25	+5V	49	50	**Ground**	**50**

Table B.3 QMM-10 (PWM Capture) Pin Interface

AC104	Signal	Board	Board	Signal	AC104
1	**Analog Ground**	**1**	**2**	**V reference (5V)**	**26**
2	**Analog In Ch 1 Hi (Servo Pot 1 Hi)**	**3**	**4**	**Analog In Ch 1 Lo (Servo Pot 1 Lo)**	**27**
3	**Analog In Ch 2 Hi (Servo Pot 2 Hi)**	**5**	**6**	**Analog In Ch 2 Lo (Servo Pot 2 Lo)**	**28**
4	**Analog In Ch 3 Hi (Servo Pot 3 Hi)**	**7**	**8**	**Analog In Ch 3 Lo (Servo Pot 3 Lo)**	**29**
5	**Analog In Ch 4 Hi (Servo Pot 4 Hi)**	**9**	**10**	**Analog In Ch 4 Lo (Servo Pot 4 Lo)**	**30**
6	**Analog In Ch 5 Hi**	**11**	**12**	**Analog In Ch 5 Lo**	**31**
7	Analog In Ch 6 Hi	13	14	Analog In Ch 6 Lo	32
8	Analog In Ch 7 Hi	15	16	Analog In Ch 7 Lo	33
9	Analog In Ch 8 Hi	17	18	Analog In Ch 8 Lo	34
10	**Analog Ground**	19	20	+15V	35
11	-15V	21	22	Digital I/O Ground	36
12	Digital I/O Ch 1	23	24	Digital I/O Ch 2	37
13	Digital I/O Ch 3	25	26	Digital I/O Ch 4	38
14	Digital I/O Ch 5	27	28	Digital I/O Ch 6	39
15	Digital I/O Ch 7	29	30	Digital I/O Ch 8	40
16	Digital I/O Ch 9	31	32	Digital I/O Ch 10	41
17	Digital I/O Ch 11	33	34	Digital I/O Ch 12	42
18	Digital I/O Ch 13	35	36	Digital I/O Ch 14	43
19	Digital I/O Ch 15	37	38	Digital I/O Ch 16	44
20	External Trigger	39	40	Digital I/O Ground	45

NOTE: Channels 21-25 and 46-50 are not connected on the AC-104 front panel

Table B.4 AIM16 (Servo Pots & Differential Pressure Sensor) Pin Interface

3. INTERRUPT ROUTINE (IRQ) ASSIGNMENT

The interrupt routine vector addresses for the new miniature computer is given in Table B.5. The IRQ assignment for the original AC-104 computer is also shown as reference should the miniature computer needs to be re-configured back to the original AC-104 configuration.

IRQ	AC-104 I/O Resource	New I/O Resource
IRQ0	ROM BIOS clock tick function	Same
IRQ1	Keyboard	Same
IRQ2	Cascaded inputs from IRQs 8 - 15	QMM-5
IRQ3	Serial Ports 2 (and 4, for ET models only)	COM2
IRQ4	Serial Ports 1 (and 3, for ET models only)	COM1
IRQ5	LPT2 or Flex/104 IP Carrier Board 3	COM4
IRQ6	Floppy drive controller	Same
IRQ7	Parallel Port or optional AX-10425 frequency driver	QMM-10
IRQ8	Watchdog timer and error handling	Same
IRQ9	Ethernet	Same
IRQ10	Controller pause function	COM3
IRQ11	Flex/104 IP Carrier Board 1	Removed
IRQ12	Not Used	Same
IRQ13	Reserved for co-processor (not used)	Same
IRQ14	IDE hard disk controller	Same
IRQ15	Flex/104 IP Carrier Board 2	Same

Table B.5 IRQ Assignment

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. SOFTWARE DRIVERS & LQR DESIGN CODE

This appendix contains the "C" source code written to receive and decode the data streams from the Ag132 GPS receiver and the Crossbow AHRS400CA-100 serving as the IMU. The IMU and GPS interface drivers were written by the author for this project while the GPS messages decoding routines were written by Dr. Vladimir Dobrokhodov for a previous project. All the C-codes had to be packaged into MATLAB's S-Function Level 2 structure which adopts a specific sequence to initialize a simulation block, update its states, control sampling rates, output data and terminate the function. Each set of code has to be "mex" by a compatible C-compiler in MATLAB and 'build' into executable code by xPC's Real-Time Workshop before it can be called from within a Simulink block as a S-Function. Details of how this is done are discussed in Refs [12] and [13].

1. CROSSBOW AHRS DATA RECEIVE DRIVER

```
/*-------------------------------------------------------------------------------- */
/*  CROSSBOW AHRS Interface Driver                                                */
/*  This routine is a modification of the rs232brec.c routine from Mathworks.     */
/*  Function:  Receive undelimited RS232 bytes from Crossbow AHRS.                */
/*             Implements a two-buffer system to collect all available data       */
/*             Search for Crossbow header byte, test if checksum of next X-1 bytes */
/*             tally with checksum byte. If so, valid Crossbow message. Output.   */
/*             If not adequate bytes to form new message, exit with last message  */
/*             X = width of Crossbow message specified by user in block's mask    */
/*  Jan 18, 2002                                                                  */
/*  Filename:  xbowrcv1p3.c                                                       */
/*  Written by:  Bock-Aeng Lim                                                    */
/*-------------------------------------------------------------------------------*/
/* Original Source file comments:
/* $Revision: 1.1 $ $Date: 2001/07/20 22:11:41 $ */
/* rs232rec.c - xPC Target, non-inlined S-function driver for RS-232 receive (asyn)  */
/* Copyright 1996-2001 The MathWorks, Inc.
*/

#define S_FUNCTION_LEVEL 2
#undef S_FUNCTION_NAME
#define S_FUNCTION_NAME xbowrcv1p3

#include <stddef.h>
#include <stdlib.h>

#include "tmwtypes.h"
#include "simstruc.h"

#ifdef MATLAB_MEX_FILE
#include "mex.h"
#else
#include <windows.h>
#include <string.h>
#include "rs232_xpcimport.h"
#include "time_xpcimport.h"
```

```
#endif

/* Input Arguments from Simulink block's user mask */
#define NUMBER_OF_ARGS          (3)
#define PORT_ARG                ssGetSFcnParam(S,0) /* COM port to use */
#define WIDTH_ARG               ssGetSFcnParam(S,1) /* max width */
#define SAMP_TIME_ARG           ssGetSFcnParam(S,2) /* User specified sample time */

#define NO_I_WORKS              (3)     /* current pos pointer in buf, rec length,
bufCount */
#define NO_R_WORKS              (0)
#define NO_P_WORKS              (0)
#define NO_D_WORKS              (1)          /* for buf array */

#define HEADER                  255          /* Crossbow message header byte  */

static char_T msg[256];
extern int rs232ports[];


static void mdlInitializeSizes(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE
#include "rs232_xpcimport.c"
#include "time_xpcimport.c"
#endif

    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        sprintf(msg,"Wrong number of input arguments passed.\n"
                "%d arguments are expected\n",NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return;
    }


    /* Set-up size information */
    ssSetNumContStates( S, 0);
    ssSetNumDiscStates( S, 0);
    ssSetNumOutputPorts(S, 2);          /* data, "done pulse" */
    ssSetNumInputPorts( S, 2);          /* rec length, enable */

    ssSetOutputPortWidth(S, 0, 1);       /* Function-call */

    ssSetOutputPortWidth(S, 1, (int)mxGetPr(WIDTH_ARG)[0]);
    ssSetOutputPortDataType(S, 1, SS_UINT8);

    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortDirectFeedThrough(S, 1, 1);
    ssSetInputPortWidth(            S, 0, 1);
    ssSetInputPortWidth(            S, 1, 1);

    ssSetInputPortRequiredContiguous(S, 0, 1);
    ssSetInputPortRequiredContiguous(S, 1, 1);

    ssSetNumSampleTimes(S,1);
    ssSetNumIWork(S, NO_I_WORKS);
    ssSetNumRWork(S, NO_R_WORKS);
    ssSetNumPWork(S, NO_P_WORKS);
    ssSetNumDWork(S, NO_D_WORKS);

    ssSetDWorkDataType(S, 0, SS_UINT8);
    ssSetDWorkWidth(   S, 0, (int)mxGetPr(WIDTH_ARG)[0]);
    ssSetDWorkWidth(   S, 0, 2048);


    ssSetNumModes(         S, 0);
    ssSetNumNonsampledZCs( S, 0);

    ssSetSFcnParamNotTunable(S,0);
    ssSetSFcnParamNotTunable(S,1);
```

```
    ssSetSFcnParamNotTunable(S,2);

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE | SS_OPTION_PLACE_ASAP);
}

/* Function to initialize sample times */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, mxGetPr(SAMP_TIME_ARG)[0]);
    if (mxGetN((SAMP_TIME_ARG))==1) {
        ssSetOffsetTime(S, 0, 0.0);
    } else {
        ssSetOffsetTime(S, 0, mxGetPr(SAMP_TIME_ARG)[1]);
    }
    ssSetCallSystemOutput(S, 0);
}

#define MDL_START  /* Change to #undef to remove function */
#if defined(MDL_START)
static void mdlStart(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE

    ssGetIWork(S)[0] = 0;     /* set current buf pointer = 0 */
    ssGetIWork(S)[2] = 0;     /* set bufCount = 0 */

#endif
}
#endif


/* Function to compute outputs */
static void mdlOutputs(SimStruct *S, int_T tid)
{
#ifndef MATLAB_MEX_FILE
    int width   = (int)mxGetPr(WIDTH_ARG)[0];          /* specify output port width   */
    int port    = (int)mxGetPr(PORT_ARG )[0] - 1; /* specify COM# */
    unsigned char tmp;                              /* temp char holder */
    unsigned char *buf = (unsigned char *)ssGetDWork(S, 0); /* uchar buffer to contain
                                                          bytes from serial port*/
    int *current   = ssGetIWork(S);/* current = addr of current position pointer in buf */
    int *recLength = ssGetIWork(S) + 1;    /* recLength = addr of receieved data length */
    int *bufCount = ssGetIWork(S)+ 2;      /* count number of useful bytes in buf. */
    int  serbufCount;         /* count number of useful bytes collected in Serial buf*/
    int  HeaderFound, i, j, bufStop, sumbytes, chksum;
    int  nextbytetoprocess, lastHeaderPos, EOB;



    if (ssGetInputPortRealSignal(S, 1)[0] == 0)      /* function is disabled. Stop
                                                     processing and get out */
    return;



    serbufCount = rl32eReceiveBufferCount(port);    /* Check number of bytes available */

    while (serbufCount) {   /* transfer everything in serial buffer to buf */
        tmp = rl32eReceiveChar(port);
        if ((tmp & 0xff00) != 0) {           /* only last 8 bits can be non-zero */
            printf("RS232Receive Error: char & 0xff00 != 0 \n");
            return;
        }

         buf[(*current)++] = tmp & 0xff;             /* put valid char into buf */
        serbufCount--;                               /* reduce serbufCount */
       (*bufCount)++;                          /* increase bufCount correspondingly */
    }


// FROM HERE, IMPLEMEMT CHECKSUM COMPUTATION & TRANSFER DECODED MESSAGE TO OUTPUT
```

```
        HeaderFound = 0;
        i = 0;
        EOB = 0;

        if (*bufCount<30) return;     // Not enough bytes to decode, output old value

        while (HeaderFound==0) {

                        /* find Header byte */
                        while (buf[i] != HEADER)      {
                                if (i < *bufCount)     i++;
                                else {EOB = 1;
                                break;}
                        } /* at exit, buf[i] = HEADER or EOB =1 */


                        if (EOB == 1) {
                                nextbytetoprocess = lastHeaderPos;
                                break;
                        }


                        if (*bufCount - i < width-1) {
                                nextbytetoprocess = i;
                                break;
                        }
                        else {
                                sumbytes = 0;                   /* Compute checksum */
                                for (j=1;j<width-1;j++) {
                                sumbytes = sumbytes + buf[i+j];
                                }
                                chksum = sumbytes % 256;

                                if  (chksum == buf[i+width-1]) {
                                  HeaderFound = 1;
                                        memcpy(ssGetOutputPortSignal(S,1),buf+i,width);
                                        nextbytetoprocess = i + width;
                                } // if checksum tally
                                else { // checksum doesn't tally
                                        lastHeaderPos = i;
                                        i++;  //skip current FF, find next FF
                                        }
                        } //else
        } /* while HeaderFound = 0*/


                if (*bufCount>100) { //BufCount large, too many old bytes, flush buffer
                        *bufCount = 0;
                        *current = *bufCount;
                }
                else {  // Pack useful bytes in buf to front of buf for next routine call
                         bufStop = *bufCount;  // bufStop = no. of bytes in pre-packed buf
                        *bufCount = *bufCount - nextbytetoprocess;  // bufCount = no. of
                                                                      bytes after packing
                                i = 0;
                                for (j=nextbytetoprocess;j<bufStop;j++) {
                                        buf[i] = buf[j];
                                        i++;
                                }
                                *current = *bufCount; // update pointer to end of buf
                }

         ssCallSystemWithTid(S, 0, 0); /* issue done pulse to outport 0 */

    return;

#endif
}

/* Function to perform housekeeping at execution termination */
```

```
static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

2. GPS DATA RECEIVE DRIVER

```
/*-------------------------------------------------------------------------------- */
/*  TRIMBLE GPS Interface Driver                                                    */
/*  This routine is a modification of the rs232brec.c routine from Mathworks.       */
/*  Function:  Receive undelimited RS232 bytes from Ag132 GPS. Search for sentence  */
/*             header '$'. Compute checksum for next 5 bytes to determine if sentence*/
/*             is 'GPGGA' or 'GPRMC'. Form message, output message at Port1 in      */
/*             X byte width, output Header Index at Port2 to identify 'GPGGA'       */
/*             or 'GPRMC' for follow-on processing.                                 */
/*             X = is max width of message specified by user in block's mask        */
/*  Original Feb 1, 2002                                                           */
/*  Revised  Mar 11, 2002                                                          */
/*  Filename:  gpsrcv.c                                                            */
/*  Written by:  Bock-Aeng Lim and Dr. Vladimir Dobohodkov                         */
/*-------------------------------------------------------------------------------- */
/* Original comments:
/* $Revision: 1.1 $ $Date: 2001/07/20 22:11:41 $ */
/* rs232rec.c - xPC Target, non-inlined S-function driver for RS-232 receive */

#define S_FUNCTION_LEVEL 2
#undef S_FUNCTION_NAME
#define S_FUNCTION_NAME gpsrcv

#include <stddef.h>
#include <stdlib.h>

#include "tmwtypes.h"
#include "simstruc.h"

#ifdef MATLAB_MEX_FILE
#include "mex.h"
#else
#include <windows.h>
#include <string.h>
#include "rs232_xpcimport.h"
#include "time_xpcimport.h"
#endif

/* Input Arguments */
#define NUMBER_OF_ARGS          (3)
#define PORT_ARG                ssGetSFcnParam(S,0)
#define WIDTH_ARG               ssGetSFcnParam(S,1) /* max width is the max length of GPS
sentence */
#define SAMP_TIME_ARG           ssGetSFcnParam(S,2)

#define NO_I_WORKS              (3)     /* current pos ptr in buf, rec length, bufCount*/
#define NO_R_WORKS              (0)
#define NO_P_WORKS              (0)
#define NO_D_WORKS              (1)          /* for buf array */

#define HEADER                               (36) // $- sign

static char_T msg[256];
extern int rs232ports[];

// unsigned char* gl_buf; global variable to save captured bytes between sample steps
```

```
static void mdlInitializeSizes(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE
#include "rs232_xpcimport.c"
#include "time_xpcimport.c"
#endif

    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        sprintf(msg,"Wrong number of input arguments passed.\n"
                "%d arguments are expected\n",NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return;
    }


    /* Set-up size information */
    ssSetNumContStates( S, 0);
    ssSetNumDiscStates( S, 0);
    ssSetNumOutputPorts(S, 3);          /* fuc-call,data, header ind */
    ssSetNumInputPorts( S, 2);          /* rec length, enable */

    ssSetOutputPortWidth(S, 0, 1);           /* Function-call */

    ssSetOutputPortWidth(S, 1, (int)mxGetPr(WIDTH_ARG)[0]);/* Data */
    ssSetOutputPortDataType(S, 1, SS_UINT8);

    ssSetOutputPortWidth(S, 2, 1);           /* Header index */
    ssSetOutputPortDataType(S, 2, SS_UINT8);

    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortDirectFeedThrough(S, 1, 1);
    ssSetInputPortWidth(            S, 0, 1);
    ssSetInputPortWidth(            S, 1, 1);

    ssSetInputPortRequiredContiguous(S, 0, 1);
    ssSetInputPortRequiredContiguous(S, 1, 1);

    ssSetNumSampleTimes(S,1);
    ssSetNumIWork(S, NO_I_WORKS);
    ssSetNumRWork(S, NO_R_WORKS);
    ssSetNumPWork(S, NO_P_WORKS);
    ssSetNumDWork(S, NO_D_WORKS);

    ssSetDWorkDataType(S, 0, SS_UINT8);
       ssSetDWorkWidth(   S, 0, (int)mxGetPr(WIDTH_ARG)[0]);
    ssSetDWorkWidth(   S, 0, 2048);


    ssSetNumModes(         S, 0);
    ssSetNumNonsampledZCs( S, 0);

    ssSetSFcnParamNotTunable(S,0);
    ssSetSFcnParamNotTunable(S,1);
    ssSetSFcnParamNotTunable(S,2);

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE | SS_OPTION_PLACE_ASAP);
}

/* Function to initialize sample times */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, mxGetPr(SAMP_TIME_ARG)[0]);
    if (mxGetN((SAMP_TIME_ARG))==1) {
        ssSetOffsetTime(S, 0, 0.0);
    } else {
        ssSetOffsetTime(S, 0, mxGetPr(SAMP_TIME_ARG)[1]);
    }
    ssSetCallSystemOutput(S, 0);
}
```

```
#define MDL_START  /* Change to #undef to remove function */
#if defined(MDL_START)
static void mdlStart(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE

    ssGetIWork(S)[0] = 0;     /* set current buf pointer = 0 */
       ssGetIWork(S)[2] = 0;  /* set bufCount = 0 */

#endif
}
#endif


/* Function to compute outputs */
static void mdlOutputs(SimStruct *S, int_T tid)
{
#ifndef MATLAB_MEX_FILE
    int width   = (int)mxGetPr(WIDTH_ARG)[0];          /* specify output port width
=WIDTH_ARG that is the max length of GPS sentence  */
    int port    = (int)mxGetPr(PORT_ARG )[0] - 1; /* specify COM# */
    unsigned char tmp;                                                  /* temp char holder
*/
    unsigned char *buf = (unsigned char *)ssGetDWork(S, 0); /* uchar buffer to contain
bytes from serial port*/
    int *current   = ssGetIWork(S);                             /* current = addr of
current position pointer in buf */
    int *recLength = ssGetIWork(S) + 1;                         /* recLength = addr of
receieved data length */
       int     *bufCount = ssGetIWork(S)+ 2;                    /* count number of useful
bytes in buf. */
    int  serbufCount;                                                   /* count number of
useful bytes collected in Serial buf*/
       int  HeaderFound, i, j, bufStop, chksum, nextbytetoprocess, lastHeaderPos, EOB;
       int  GGA=358,RMC=377;// checksum of 'GPGGA','GPRMC' sentences's header
       int* bl_header;// boolean values for GGA=1 and RMC=2 sentences, 0=nothing found
       int  headwidth=5;// length of GPS header except '$'


    if (ssGetInputPortRealSignal(S, 1)[0] == 0)             /* function is disabled. Stop
processing and get out */
        return;

       serbufCount = rl32eReceiveBufferCount(port);    /* Check number of bytes available
*/

       while (serbufCount)
       {   /* transfer everything in serial buffer to buf */
        tmp = rl32eReceiveChar(port);
        if ((tmp & 0xff00) != 0)
        {                       /* only last 8 bits can be non-zero */
            printf("RS232Receive Error: char & 0xff00 != 0 \n");
            return;
        }
              buf[(*current)++] = tmp & 0xff;              /* put valid char into buf */
        serbufCount--;                                      /* reduce serbufCount */
              (*bufCount)++;                                                /* increase
bufCount correspondingly */
    }
       /*Initialize logical flags*/
       HeaderFound = 0;
       i = 0;
       EOB = 0;//end of buffer

       if (*bufCount<width) return;  // Not enough bytes to decode, output old value

       while (HeaderFound==0)
       {/* find Header byte = '$'=36 */
       while (buf[i] != HEADER)
               {
                          if (i < *bufCount) i++;
```

```
                    else
                        {
                            EOB = 1; break;
                         }
          } /* at exit, buf[i] = HEADER or EOB =1 */

   if (EOB == 1)
       {   nextbytetoprocess = lastHeaderPos;
          break;
        }//end of if

   if (*bufCount - i < width-1)
          {   nextbytetoprocess = i;
          break;
          }
   else
          {chksum = 0;                          /* Compute checksum */
          for (j=1;j<headwidth+1;j++)
          {
              chksum = chksum + buf[i+j];
           }
          }// calculate "checksum" of GPS header={$GPGGA,$GPRMC}
                 if  (chksum == GGA)
                 { HeaderFound = 1;
                   *bl_header=1;
                   ggalng=0;
                   while(*(buf+i+headwidth+1+ggalng) != 13) ggalng++; //"0D"=13,
                   memcpy(ssGetOutputPortSignal(S,1),buf+i+headwidth+1,ggalng);
                   nextbytetoprocess = i + ggalng;
                  } // if checksum tally
                  else
                  {
                   if (chksum == RMC)
                         {  HeaderFound = 1;
                            *bl_header=2;
                             rmclng = 0;
                             while(*(buf+i+headwidth+1+rmclng) != 13) rmclng++;
                 memcpy(ssGetOutputPortSignal(S,1),buf+i+headwidth+1,rmclng);
                            nextbytetoprocess = i + rmclng;
                         } // if checksum tally
                         else
                         { // checksum doesn't tally
                                 lastHeaderPos = i;
                                 *bl_header=0;
                          }// end of checksum searching
                 }

                 memcpy(ssGetOutputPortSignal(S,2),bl_header,1);

    } /* while HeaderFound = 0*/

          if (*bufCount>300)
          { // If bufCount too large, too many old bytes, just flush buffer
                 *bufCount = 0;
                 *current = *bufCount;
          }
          else {  // Pack useful bytes in buf to front of buf.
                   bufStop = *bufCount;  // bufStop = no. of bytes in pre-packed buf
                   *bufCount = *bufCount - nextbytetoprocess;  // bufCount = no. of
                                                                bytes after packing
                 i = 0;
                 for (j=nextbytetoprocess;j<bufStop;j++)
                  {
                        buf[i] = buf[j];
                        i++;
                  }
                  *current = *bufCount; // update pointer to end of buf
                 }

   ssCallSystemWithTid(S, 0, 0); /* issue done pulse to outport 0 */
return;
```

```
#endif
}

/* Function to perform housekeeping at execution termination */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

3. GPS GPGGA MESSAGE DECODER

```
/*--------------------------------------------------------------------------------- */
/*  GPGGA MESSAGE DECODING ROUTINE                                                  */
/*  Function:  This routine decodes the output from gpsrcv block and decode the     */
/*             GPGGA message.                                                       */
/*  Feb 1, 2002                                                                    */
/*  Filename:  gpgga.c                                                             */
/*  Written by:  Vladimir Dobohodkov                                               */
/*--------------------------------------------------------------------------------- */
/*
 * File : gpgga.c
 * $Revision: 1.00 $V.Dobrokhodov
 */

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>

#define S_FUNCTION_NAME  gpgga
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

/* Input Arguments */
#define NUMBER_OF_ARGS          (1)
#define WIDTH               ssGetSFcnParam(S,0) /* WIDTH is the max length of incoming
GPS sentence */

/*================*
 * Build checking *
 *================*/
static char_T msg[256];


/* Function: mdlInitializeSizes ===============================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
        ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        sprintf(msg,"Wrong number of input arguments passed.\n"
                "%d arguments are expected\n",NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return; /* Parameter mismatch will be reported by Simulink */
    }

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, (int)mxGetPr(WIDTH)[0]);//DYNAMICALLY_SIZED
```

```
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, 14);//14DYNAMICALLY_SIZED

    ssSetNumSampleTimes(S, 1);

    /* Take care when specifying exception free code - see sfuntmpl_doc.c */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_USE_TLC_WITH_ACCELERATOR);
}


/* Function: mdlInitializeSampleTimes =========================================
 * Abstract:
 *    Specifiy that we inherit our sample time from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

int gpssmbl(int_T* input,int length)
{
        char tmp[]="\0",output[]="\0";
        int i,gpstmp;
        for (i=0;i<length;i++)
        {       tmp[0]=(char)(*input);strncat(output,tmp,1);input++;}
        switch (output[0])
        {       case 'N': gpstmp=0;break;/*North*/
                case 'E': gpstmp=1;break;/*East*/
                case 'S': gpstmp=2;break;/*South*/
                case 'W': gpstmp=3;break;/*West*/
                case 'A': gpstmp=4;break;/*Valid or Autonomous*/
                case 'D': gpstmp=5;break;/*Differential*/
                case 'V': gpstmp=6;break;/*Non-Valid*/
                case 'M': gpstmp=7;break;/*Meters*/
                default: gpstmp=6;break;/*Non-Valid*/
        }/*end of switch*/
        return gpstmp;
};// end of gpssmbl

/* Function: bin2ascii ========================================================
  Abstract: function provides GPS data convertion to ASCII and then to FLOAT
representations */
void bin2ascii(int_T* in,int length, char* ext)
{       int* input=in;
        char tmp[]="\0";
        int i;
        for (i=0;i<length;i++)
        {
                tmp[0]=(char)(*input);
                strncat(ext,tmp,1);
                input++;
        }
};// end of bin2ascii



/* Function: mdlOutputs =======================================================
 * Abstract:
 *
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T             i=0,j=0;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);/* Incoming data stream*/
     real_T            *y    = ssGetOutputPortRealSignal(S,0);
    /*int_T             width = ssGetOutputPortWidth(S,0);*/
```

```
/*************************************************************************/
      int_T tempbuf[100]; //here we make an aliase for the uPtrs
      char ext[]="\0";
      int count=0,len_in;
      double sys[20];//output array of decoded data
      double res[1]={0}; //output piece of decoded data
      real_T tmp;
/*************************************************************************/
     for (i=0; i<(*uPtrs[0]); i++){tempbuf[i] = ( int_T)(*uPtrs[i+1]);}
/*************************************************************************/

    //GPGGA sentence of GPS message
    i=0;
    if ((real_T) tempbuf[i] != (real_T) 44)
    {  /*printf("\n Error! 44 expected, received %d",*tempbuf);*/
       return;}
    else count++; /*miss first comma sign and define shift*/

    /* UTC & Latitude -1,2*/
    for(i=0;i<2;i++)
    {
    len_in=0;/*initialize it again*/
    while (tempbuf[count+len_in] != 44)
    {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
    bin2ascii((tempbuf+count),len_in,ext);
    *y++=(real_T)atof(ext);
    *ext=NULL;/*initialize it again*/
    count+=len_in+1;//miss next comma sign and define new shift
    }/*end of for*/


    /*Direction of latitude -3*/
    len_in=0;/*initialize it again*/
    while (tempbuf[count+len_in] != 44)
    {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
    *y++=gpssmbl((tempbuf+count),len_in);
    count+=len_in+1;//miss next comma sign and define new shift

    /*Longitude -4*/
    len_in=0;/*initialize it again*/
    while (tempbuf[count+len_in] != 44)
    {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
    bin2ascii((tempbuf+count),len_in,ext);
    *y++=(real_T)atof(ext);
    *ext=NULL;/*initialize it again*/
    count+=len_in+1;//miss next comma sign and define new shift

    /*Direction of longitude -5*/
    len_in=0;/*initialize it again*/
    while (tempbuf[count+len_in] != 44)
    {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
    *y++=gpssmbl((tempbuf+count),len_in);
    count+=len_in+1;//miss next comma sign and define new shift

    /*GPS quality indicator -6;
    Number of SVs            -7;
    HDOP                          -8;
    Antenna height           -9*/

    for(i=0;i<4;i++)
    {
    len_in=0;/*initialize it again*/
    while (tempbuf[count+len_in] != 44)
    {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
    bin2ascii((tempbuf+count),len_in,ext);
    *y++=(real_T)atof(ext);
```

```
     *ext=NULL;/*initialize it again*/
     count+=len_in+1;//miss next comma sign and define new shift
     }/* end of for*/

     /*Altitude in meters -10*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     *y++=gpssmbl((tempbuf+count),len_in);
     count+=len_in+1;//miss next comma sign and define new shift

     /*Geoidal separation -11*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     bin2ascii((tempbuf+count),len_in,ext);
     *y++=(real_T)atof(ext);
     *ext=NULL;/*initialize it again*/
     count+=len_in+1;//miss next comma sign and define new shift

     /*Geoidal separation in meters -12*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     *y++=gpssmbl((tempbuf+count),len_in);
     count+=len_in+1;//miss next comma sign and define new shift

     /*Age of DGPS data -13*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     bin2ascii((tempbuf+count),len_in,ext);
     *y++=(real_T)atof(ext);
     *ext=NULL;/*initialize it again*/
     count+=len_in+1;//miss next comma sign and define new shift

     /*Base station ID-14*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 42)/*42 ='*' It's the beginning of next gps
message*/
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     bin2ascii((tempbuf+count),len_in,ext);
     *y++=(real_T)atof(ext);
     *ext=NULL;/*initialize it again*/
     count+=len_in+1;//miss next comma sign and define new shift
 /*END of DGPS GGA sentence*/
/*****************************************************************************/
}


/* Function: mdlTerminate =================================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
}


#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

4. GPS GPRMC MESSAGE DECODER

```
/*-------------------------------------------------------------------------------- */
/*  GPRMC MESSAGE DECODING ROUTINE                                                 */
/*  Function:  This routine decodes the output from gpsrcv block and decode the    */
/*             GPRMC message.                                                      */
/*  Feb 1, 2002                                                                    */
/*  Filename:  gprmc.c                                                             */
/*  Written by:  Vladimir Dobohodkov                                               */
/*-------------------------------------------------------------------------------- */
/*
 * File : gprmc.c
 * $Revision: 1.00 $V.Dobrokhodov
 */

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>


#define S_FUNCTION_NAME  gprmc
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

/* Input Arguments */
#define NUMBER_OF_ARGS          (1)
#define WIDTH               ssGetSFcnParam(S,0) /* WIDTH is the max length of incoming
GPS sentence */

/*================*
 * Build checking *
 *================*/
static char_T msg[256];

/* Function: mdlInitializeSizes ===============================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{

        ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        sprintf(msg,"Wrong number of input arguments passed.\n"
                "%d arguments are expected\n",NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return; /* Parameter mismatch will be reported by Simulink */
    }

        if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, (int)mxGetPr(WIDTH)[0]);//DYNAMICALLY_SIZED
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0,12 );//12=RMC

    ssSetNumSampleTimes(S, 1);

    /* Take care when specifying exception free code - see sfuntmpl_doc.c */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_USE_TLC_WITH_ACCELERATOR);
}


/* Function: mdlInitializeSampleTimes =========================================
 * Abstract:
```

```
 *    Specifiy that we inherit our sample time from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
/* Function: gps_smbl =======================================================
 * Abstract: function provides GPS symbol informatin processing */


int gps_smbl(int_T* input,int length)
{
        char tmp[]="\0",output[]="\0";
        int i,gpstmp;
        for (i=0;i<length;i++)
        {       tmp[0]=(char)(*input);strncat(output,tmp,1);input++;}
        switch (output[0])
        {       case 'N': gpstmp=0;break;/*North*/
                case 'E': gpstmp=1;break;/*East*/
                case 'S': gpstmp=2;break;/*South*/
                case 'W': gpstmp=3;break;/*West*/
                case 'A': gpstmp=4;break;/*Valid or Autonomous*/
                case 'D': gpstmp=5;break;/*Differential*/
                case 'V': gpstmp=6;break;/*Non-Valid*/
                case 'M': gpstmp=7;break;/*Meters*/
                default: gpstmp=6;break;/*Non-Valid*/
        }/*end of switch*/
        return gpstmp;
};// end of gps_smbl

/* Function: bin2_ascii =====================================================
  Abstract: function provides GPS data convertion to ASCII and then to FLOAT
representations */
void bin2_ascii(int_T* in,int length, char* ext)
{       int* input=in;
        char tmp[]="\0";
        int i;
        for (i=0;i<length;i++)
        {
                tmp[0]=(char)(*input);
                strncat(ext,tmp,1);
                input++;
        }
};// end of bin2_ascii



/* Function: mdlOutputs =====================================================
 * Abstract:
 *
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T             i=0,j=0;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);/* Incoming data stream*/
     real_T            *y    = ssGetOutputPortRealSignal(S,0);
    /*int_T             width = ssGetOutputPortWidth(S,0);*/

 /***********************Interface for C++ programm*********************************/
        int_T tempbuf[100]; //here we make an aliase for the uPtrs
        char ext[]="\0";
        int count=0,len_in;
        double sys[20];//output array of decoded data
        double res[1]={0}; //output piece of decoded data
        real_T tmp;
 /***********************Merge data to temporary array**************************/
      for (i=0; i<(*uPtrs[0]); i++){tempbuf[i] = ( int_T)(*uPtrs[i+1]);}
 /******************************************************************************/

     //GPRMC sentence of GPS message
```

```
     i=0;
     if ((real_T) tempbuf[i] != (real_T) 44) return;
     else count++; /*miss first comma sign and define shift*/

     /* UTC -1*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     bin2_ascii((tempbuf+count),len_in,ext);
     *y++=(real_T)atof(ext);
     *ext=NULL;/*initialize it again*/
     count+=len_in+1;//miss next comma sign and define new shift


     /*Status -2*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     *y++=gps_smbl((tempbuf+count),len_in);
     count+=len_in+1;//miss next comma sign and define new shift

     /*Latitude -3*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     bin2_ascii((tempbuf+count),len_in,ext);
     *y++=(real_T)atof(ext);
     *ext=NULL;/*initialize it again*/
     count+=len_in+1;//miss next comma sign and define new shift

     /*Latitude direction -4*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     *y++=gps_smbl((tempbuf+count),len_in);
     count+=len_in+1;//miss next comma sign and define new shift

        /*Longitude -5*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     bin2_ascii((tempbuf+count),len_in,ext);
     *y++=(real_T)atof(ext);
     *ext=NULL;/*initialize it again*/
     count+=len_in+1;//miss next comma sign and define new shift

     /*Direction of logitude -6*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     *y++=gps_smbl((tempbuf+count),len_in);
     count+=len_in+1;//miss next comma sign and define new shift

     /*Speed over ground[knots] -7*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     bin2_ascii((tempbuf+count),len_in,ext);
     *y++=(real_T)atof(ext);
     *ext=NULL;/*initialize it again*/
     count+=len_in+1;//miss next comma sign and define new shift

     /*Track made good,True[degree] -8*/
     len_in=0;/*initialize it again*/
```

```
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     bin2_ascii((tempbuf+count),len_in,ext);
     *y++=(real_T)atof(ext);
     *ext=NULL;/*initialize it again*/
     count+=len_in+1;//miss next comma sign and define new shift

         /*Date in  dd/mm/yy                  -9;
     Manetic variation [degree]              -10;*/

     for(i=0;i<2;i++)
     {
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     bin2_ascii((tempbuf+count),len_in,ext);
     *y++=(real_T)atof(ext);
     *ext=NULL;/*initialize it again*/
     count+=len_in+1;//miss next comma sign and define new shift
     }/* end of for*/

     /*Direction of Magnetic variation-11*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 44)
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     *y++=gps_smbl((tempbuf+count),len_in);
     count+=len_in+1;//miss next comma sign and define new shift


     /*Mode indicator(A(4)-autonomous;D(5)-differencial;N(0)-not valid) -12*/
     len_in=0;/*initialize it again*/
     while (tempbuf[count+len_in] != 42)/*=42*/
     {++len_in;}//end of while to count the length of GPS field - looking for next
","{44-ASCII}
     *y++=gps_smbl((tempbuf+count),len_in);
     count+=len_in+1;//miss next comma sign and define new shift

 /*END of DGPS RMC sentence*/
/*************************************************************************/
}

/* Function: mdlTerminate =====================================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
}



#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

5. MATLAB CODE FOR LQR CONTROLLER DESIGN

```
close all;
clear all;
clc;

load frogabcd10;        % load plant model
i = 1;

 rank(ctrb(as,bs))                    %  check that (A,B) is controllable and
 rank(obsv(as,cs'*cs))        %       (A,Q) is observable. Criteria for LQR. Done only
once.

%% Design parameters for tzeros in synthesis.mdl
xi_h = 0.9;
wn_h = 0.4;
xi_v = 0.9;
wn_v = 1.2;
pole_vt = 0.7;
pole_psi = 1.5;

% Design parameters for LQR
q1 = diag([1, 1, 1, 1]);              % increase qii, to increase cmd loop bw
r1 = diag([2000,5000,1500,2]);          % decrease rii, to increase control loop bw

%% Obtaining the synthesis model
disp(' ');
disp('tranmission zeros of plant with Vt,h,v,psi output');
tzero(as,bs,cs,ds)
[as1,bs1,cs1,ds1] = linmod('synthesis'); %,'v5');
disp('transmssion zeros of synthesis model');
damp(tzero(as1,bs1,cs1,ds1))

synth = [1 2*xi_h*wn_h wn_h^2];
syntv = [1 2*xi_v*wn_v wn_v^2];
disp('Synthesis zeros are:')
damp(roots(synth));
damp(roots(syntv));
disp(-pole_vt);
disp(-pole_psi);

%% Computing the feedback gains
[k,p,e] = lqr(as1,bs1,cs1'*q1*cs1,r1);


%% Obtaining the closed-loop system
kp = k(:,1:10);
ki = k(:,11:14);
[ac,bc,cc,dc] = linmod('model10'); %,'v5');


%% Check criteria 1:  Feedback system must be stable
disp('Ensure that all the closed loop poles are stable');
damp(eig(ac))

%% Now include actuator into model to see time-response
[ac1,bc1,cc1,dc1] = linmod('model10wAct'); %,'v5');
damp(eig(ac1))

% plots Vt_out to step input in Vt_cmd

figure(i); i=i+1;
step(ac1,bc1(:,5),cc1(5,:),dc1(5,5));
% axis([0 20 -0.2 1.8])
title('Vt out to step input in Vt cmd');

% plots h_out to step input in h_cmd
figure(i); i=i+1;
step( ac1,bc1(:,6),cc1(6,:),dc1(6,6) );
```

```
% axis([0 20 -0.2 1.8])
title('h out to step input in h cmd');
grid;

% plots v_out to step input in v_cmd
figure(i); i=i+1;
step( ac1,bc1(:,7),cc1(7,:),dc1(7,7) );
% axis([0 20 -0.2 1.8])
title('v out to step input in v cmd');
grid;

% plots psi_out to step input in psi_cmd
figure(i); i=i+1;
step( ac1,bc1(:,8),cc1(8,:),dc1(8,8) );
% axis([0 20 -0.2 1.8])
title('psi out to step input in psi cmd');
grid;


%% Criteria 5:  Aileron, Elevator, Rudder loop bandwidth < 10 rad/s
%%                                    Thrust loop bandwidth < 5 rad/s

figure(i); i=i+1;
margin( ac1,bc1(:,1),cc1(1,:),dc1(1,1) )
title('Bode of closed loop from del aileron to aileron out. (BW ~ 10 rad/s)');

figure(i); i=i+1;
margin( ac1,bc1(:,2),cc1(2,:),dc1(2,2) )
title('Bode of closed loop from del elevator to elevator out. (BW ~ 10 rad/s)');

figure(i); i=i+1;
margin( ac1,bc1(:,3),cc1(3,:),dc1(3,3) )
title('Bode of closed loop from del rudder to rudder out. (BW ~ 10 rad/s)');

figure(i); i=i+1;
margin( ac1,bc1(:,4),cc1(4,:),dc1(4,4) )
title('Bode of closed loop from del thrust to thrust out. (BW ~ 5 rad/s)');


%% Criteria 4:  Gain margin in elevator and thrust loops should
%% be at least 6 db and phase margin 45 degrees.

[ao1,bo1,co1,do1] = linmod('open_a');
figure(i); i=i+1;
margin(ao1,bo1(:,1),co1(1,:),do1(1,1))
title('Bode of open loop from del aileron cmd to aileron out');
disp('Gain and phase margin for aileron loop');

[ao2,bo2,co2,do2] = linmod('open_e');
figure(i); i=i+1;
margin(ao2,bo2(:,2),co2(2,:),do2(2,2))
title('Bode of open loop from del elevator cmd to elevator out');
disp('Gain and phase margin for elevator loop');


[ao3,bo3,co3,do3] = linmod('open_r');
figure(i); i=i+1;
margin(ao3,bo3(:,3),co3(3,:),do3(3,3))
title('Bode of open loop from del rudder cmd to rudder out');
disp('Gain and phase margin for rudder loop');


[ao4,bo4,co4,do4] = linmod('open_th');
figure(i); i=i+1;
margin(ao4,bo4(:,4),co4(4,:),do4(4,4))
title('Bode of open loop from del thrust cmd to thrust out');
disp('Gain and phase margin for thrust loop');

break;
```

LIST OF REFERENCES

1. Kaminer,I., Hespanha, J., Yakimenko, O., Proposal for study on "Distributed Adaptive Architectures for Intelligent Sensor Fusion over Dynamic (Wireless) Networks", Oct 2000.

2. Froncillo, S., "Design of Digital Control Algorithms for UAV," Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1998.

3. Hallberg, E., "On Integrated Plant, Control and Guidance Design," Ph.D Dissertation, Naval Postgraduate School, Monterey, CA, September 1997.

4. Flood, C., "Design and Evaluation of a Digital Flight Control System for the FROG Unmanned Aerial Vehicle," Engineer's Thesis, Naval Postgraduate School, Monterey, CA, Sep 2001.

5. Flood, C. & Lim, B.A., "Alternate Servo Control for FROG UAV," AA4642 Avionics II Class Project Report, Sep 2001.

6. Papageorgio, E., "Development of a Dynamic Model for a UAV," Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1997.

7. Blakelock, J.H., "Automatic control of aircraft and missiles," New York, Wiley, 1965.

8. Blight, J., "Integral LQG Design Or: Modern Control Without Math", Lecture, Boeing Military Airplane Company.

9. Kaminer, I. et al., "A Velocity Algorithm for the Implementation of Gain-scheduled Controllers," Automatica, Vol. 31, No. 8, 1995.

10. Schmidt, L. V., "Introduction to Aircraft Flight Dynamics," AIAA Education Series, 1998.

11. Yakimenko, O. et al., "Unmanned Aircraft Navigation For Shipboard Landing Using Infrared Vision," accepted for publication in IEEE Transactions of Aerospace and Electronics, 2002.

12. "MATLAB External Interfaces Version 6," The MathWorks, Inc., Nov 2000.

13. "MATLAB Writing S-Functions Version 4," The MathWorks, Inc., Nov 2000.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Prof. Isaac Kaminer, Dept. of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, California

4. Prof. Oleg Yakimenko, Dept. of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, California

5. Prof. Max Platzer, Dept. of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, California

6. Dr. Vladimir Dobrokhodov, Dept. of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, California

7. W. J. Lentz, Dept. of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, California